adafruit learning system

CircuitPython Essentials

Created by Kattni Rembor

Last updated on 2020-04-02 02:45:34 PM EDT

CircuitPython Essentials

You've gone through the Welcome to CircuitPython guide (https://adafru.it/cpy-welcome). You've already gotten everything setup, and you've gotten CircuitPython running. Great! Now what? CircuitPython Essentials!

There are a number of core modules built into CircuitPython and commonly used libraries available. This guide will introduce you to these and show you an example of how to use each one.

Each section will present you with a piece of code designed to work with different boards, and explain how to use the code with each board. These examples work with any board designed for CircuitPython, including **Circuit Playground Express**, **Trinket M0**, **Gemma M0**, **ItsyBitsy M0 Express**, **ItsyBitsy M4 Express, Feather M0 Express, Feather M4 Express, Metro M4 Express, Metro M0 Express, Trellis M4 Express, and Grand Central M4 Express**.

Some examples require external components, such as switches or sensors. You'll find wiring diagrams where applicable to show you how to wire up the necessary components to work with each example.

Let's get started learning the CircuitPython Essentials!

CircuitPython Built-Ins

CircuitPython comes 'with the kitchen sink' - *a lot* of the things you know and love about classic Python 3 (sometimes called CPython) already work. There are a few things that don't but we'll try to keep this list updated as we add more capabilities!

This is not an exhaustive list! It's simply some of the many features you can use.

Thing That Are Built In and Work

Flow Control

All the usual `if`, `elif`, `else`, `for`, `while` work just as expected.

Math

`import math` will give you a range of handy mathematical functions.

```
>>> dir(math)
['__name__', 'e', 'pi', 'sqrt', 'pow', 'exp', 'log', 'cos', 'sin', 'tan', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'copysign', 'fabs',
'floor', 'fmod', 'frexp', 'ldexp', 'modf', 'isfinite', 'isinf', 'isnan', 'trunc', 'radians', 'degrees']
```

CircuitPython supports 30-bit wide floating point values so you can use `int` and `float` whenever you expect.

Tuples, Lists, Arrays, and Dictionaries

You can organize data in `()`, `[]`, and `{}` including strings, objects, floats, etc.

Classes, Objects and Functions

We use objects and functions extensively in our libraries so check out one of our many examples like this MCP9808 library (https://adafru.it/BfQ) for class examples.

Lambdas

Yep! You can create function-functions with `lambda` just the way you like em:

```
>>> g = lambda x: x**2
>>> g(8)
64
```

Random Numbers

To obtain random numbers:

`import random`

`random.random()` will give a floating point number from `0` to `1.0`.

`random.randint(min, max)` will give you an integer number between `min` and `max`.

CircuitPython Digital In & Out

The first part of interfacing with hardware is being able to manage digital inputs and outputs. With CircuitPython, it's super easy!

This example shows how to use both a digital input and output. You can use a switch *input* with pullup resistor (built in) to control a digital *output* - the built in red LED.

Copy and paste the code into **code.py** using your favorite editor, and save the file to run the demo.

```
# CircuitPython IO demo #1 - General Purpose I/O
import time
import board
from digitalio import DigitalInOut, Direction, Pull

led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT

# For Gemma M0, Trinket M0, Metro M0 Express, ItsyBitsy M0 Express, Itsy M4 Express
switch = DigitalInOut(board.D2)
# switch = DigitalInOut(board.D5)  # For Feather M0 Express, Feather M4 Express
# switch = DigitalInOut(board.D7)  # For Circuit Playground Express
switch.direction = Direction.INPUT
switch.pull = Pull.UP

while True:
    # We could also do "led.value = not switch.value"!
    if switch.value:
        led.value = False
    else:
        led.value = True

    time.sleep(0.01)  # debounce delay
```

Note that we made the code a little less "Pythonic" than necessary. The `if/else` block could be replaced with a simple `led.value = not switch.value` but we wanted to make it super clear how to test the inputs. The interpreter will read the digital input when it evaluates `switch.value`.

For **Gemma M0, Trinket M0, Metro M0 Express, Metro M4 Express, ItsyBitsy M0 Express, ItsyBitsy M4 Express**, no changes to the initial example are needed.

Note: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

For **Feather M0 Express and Feather M4 Express**, comment out `switch = DigitalInOut(board.D2)` (and/or `switch = DigitalInOut(board.D7)` depending on what changes you already made), and uncomment `switch = DigitalInOut(board.D5)`.

For **Circuit Playground Express**, you'll need to comment out `switch = DigitalInOut(board.D2)` (and/or `switch = DigitalInOut(board.D5)` depending on what changes you already made), and uncomment `switch =`

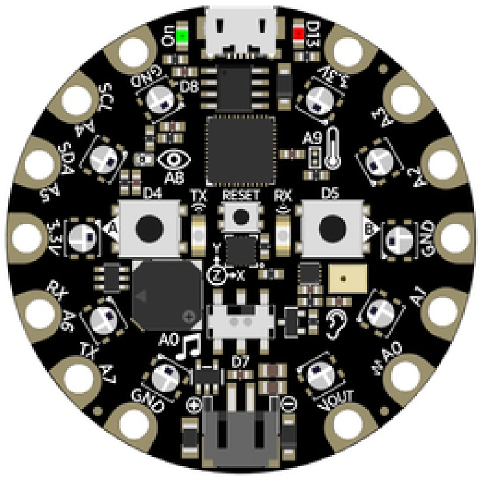
`DigitalInOut(board.D7)` .

To find the pin or pad suggested in the code, see the list below. For the boards that require wiring, wire up a switch (also known as a tactile switch, button or push-button), following the diagram for guidance. Press or slide the switch, and the onboard red LED will turn on and off.

Note that on the M0/SAMD based CircuitPython boards, at least, you can also have internal pulldowns with **Pull.DOWN** and if you want to turn off the pullup/pulldown just assign **switch.pull** = **None.**

Find the pins!

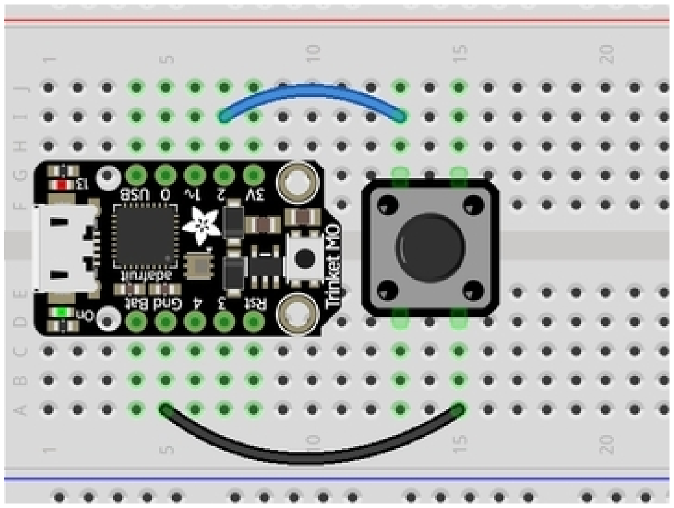
The list below shows each board, explains the location of the Digital pin suggested for use as input, and the location of the D13 LED.



Circuit Playground Express

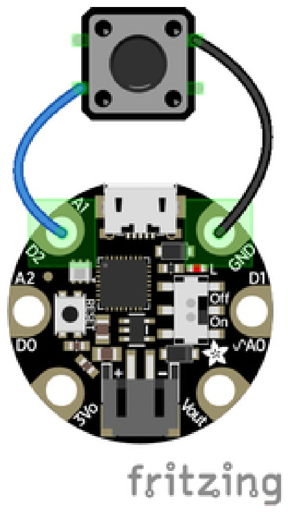
We're going to use the switch, which is pin D7, and is located between the battery connector and the reset switch on the board. D13 is labeled D13 and is located next to the USB micro port.

To use D7, comment out the current pin setup line, and uncomment the line labeled for Circuit Playground Express. See the details above!



Trinket M0

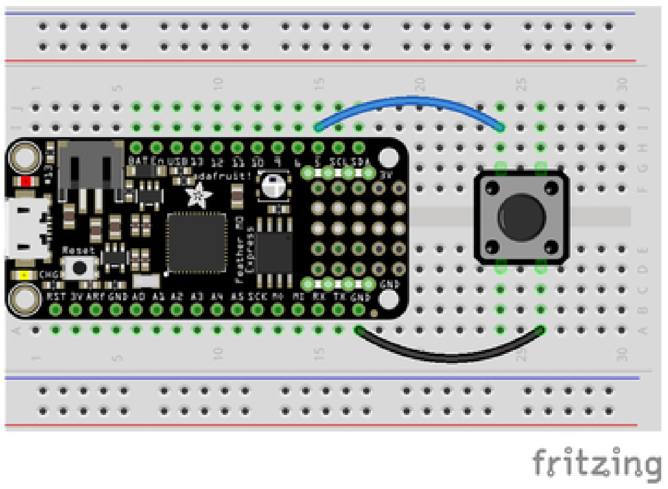
D2 is connected to the blue wire, labeled "2", and located between "3V" and "1" on the board. D13 is labeled "13" and is located next to the USB micro port.



Gemma M0

D2 is an alligator-clip-friendly pad labeled both "D2" and "A1", shown connected to the blue wire, and is next to the USB micro port. D13 is located next to the "GND" label on the board, above the "On/Off" switch.

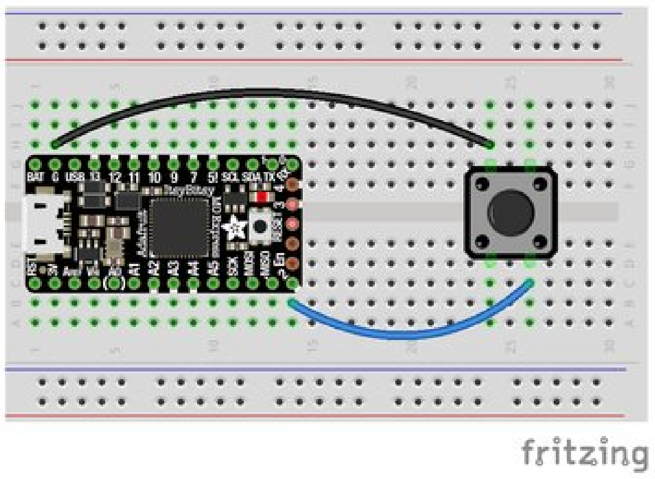
Use alligator clips to connect your switch to your Gemma M0!



Feather M0 Express and Feather M4 Express

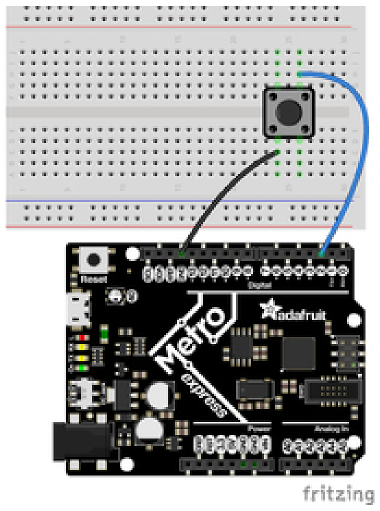
D5 is labeled "5" and connected to the blue wire on the board. D13 is labeled "#13" and is located next to the USB micro port.

To use D5, comment out the current pin setup line, and uncomment the line labeled for Feather M0 Express. See the details above!



ItsyBitsy M0 Express and ItsyBitsy M4 Express

D2 is labeled "2", located between the "MISO" and "EN" labels, and is connected to the blue wire on the board. D13 is located next to the reset button between the "3" and "4" labels on the board.



Metro M0 Express and Metro M4 Express

D2 is located near the top left corner, and is connected to the blue wire. D13 is labeled "L" and is located next to the USB micro port.

Read the Docs

For a more in-depth look at what `digitalio` can do, check out the `DigitalInOut` page in Read the Docs (https://adafru.it/C4c).

CircuitPython Analog In

This example shows you how you can read the analog voltage on the A1 pin on your board.

Copy and paste the code into **code.py** using your favorite editor, and save the file to run the demo.

```
# CircuitPython AnalogIn Demo
import time
import board
from analogio import AnalogIn

analog_in = AnalogIn(board.A1)


def get_voltage(pin):
    return (pin.value * 3.3) / 65536


while True:
    print((get_voltage(analog_in),))
    time.sleep(0.1)
```

Make sure you're running the latest CircuitPython! f you are not, you may run into an error: "AttributeError: 'module' object has no attribute 'A1'". If you receive this error, first make sure you're running the latest version of CircuitPython!

Creating the analog input

`analog1in = AnalogIn(board.A1)`

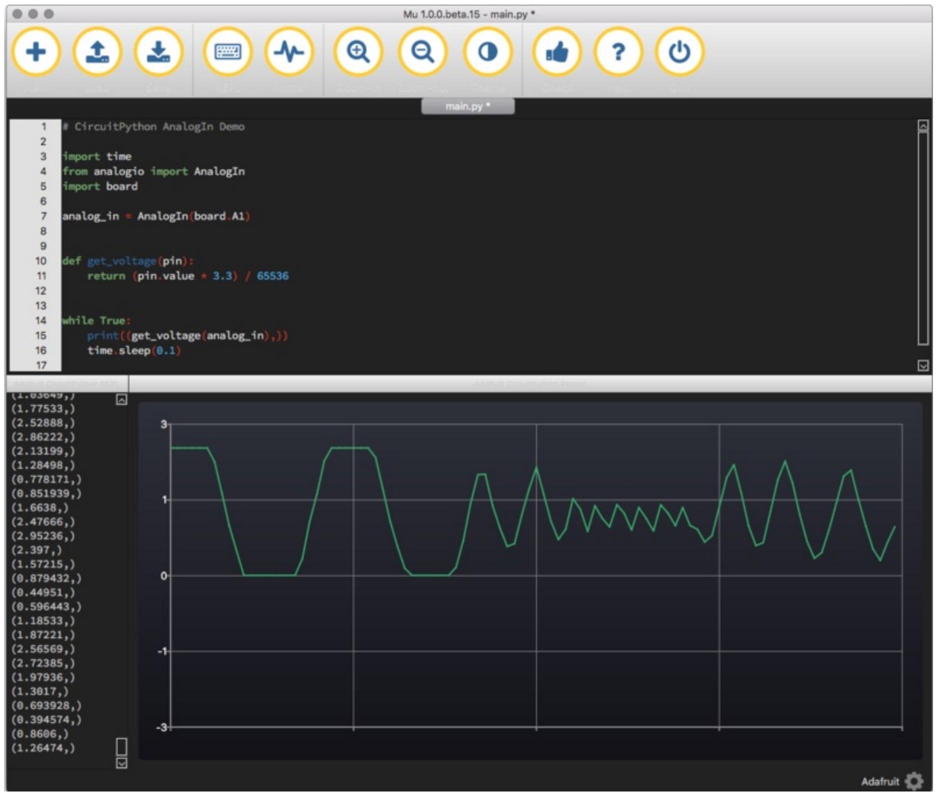
Creates an object and connects the object to A1 as an analog input.

`get_voltage` Helper

`getVoltage(pin)` is our little helper program. By default, analog readings will range from 0 (minimum) to 65535 (maximum). This helper will convert the 0-65535 reading from `pin.value` and convert it a 0-3.3V voltage reading.

Main Loop

The main loop is simple. It `prints` out the voltage as floating point values by calling `get_voltage` on our analog object. Connect to the serial console to see the results.



```python
# CircuitPython AnalogIn Demo

import time
from analogio import AnalogIn
import board

analog_in = AnalogIn(board.A1)


def get_voltage(pin):
    return (pin.value * 3.3) / 65536


while True:
    print((get_voltage(analog_in),))
    time.sleep(0.1)
```

```
(1.03649,)
(1.77533,)
(2.52888,)
(2.86222,)
(2.13199,)
(1.28498,)
(0.778171,)
(0.851939,)
(1.6638,)
(2.47666,)
(2.95236,)
(2.397,)
(1.57215,)
(0.879432,)
(0.44951,)
(0.596443,)
(1.18533,)
(1.87221,)
(2.56569,)
(2.72385,)
(1.97936,)
(1.3017,)
(0.693928,)
(0.394574,)
(0.8606,)
(1.26474,)
```

Changing It Up

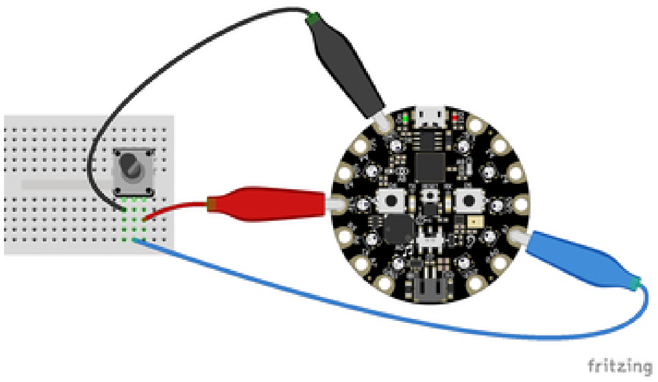
By default the pins are *floating* so the voltages will vary. While connected to the serial console, try touching a wire from **A1** to the **GND** pin or **3Vo** pin to see the voltage change.

You can also add a potentiometer to control the voltage changes. From the potentiometer to the board, connect the **left pin** to **ground**, the **middle pin** to **A1**, and the **right pin** to **3V**. If you're using Mu editor, you can see the changes as you rotate the potentiometer on the plotter like in the image above! (Click the Plotter icon at the top of the window to open the plotter.)

When you turn the knob of the potentiometer, the wiper rotates left and right, increasing or decreasing the resistance. This, in turn, changes the analog voltage level that will be read by your board on A1.

Wire it up

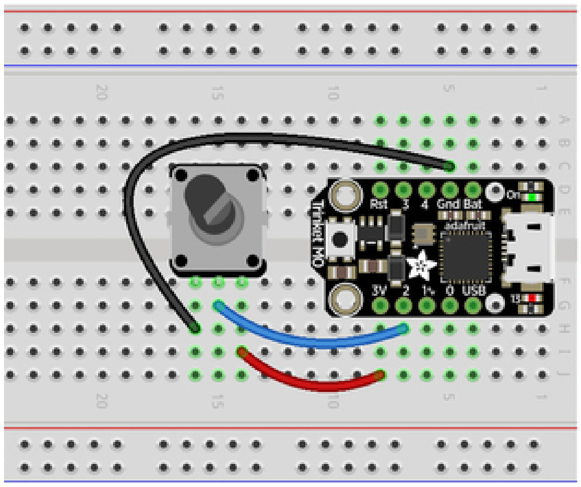
The list below shows wiring diagrams to help find the correct pins and wire up the potentiometer, and provides more information about analog pins on your board!



Circuit Playground Express

A1 is located on the right side of the board. There are multiple ground and 3V pads (pins).

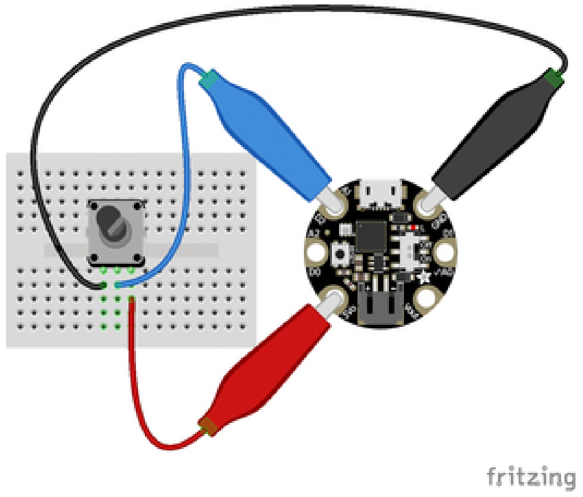
Your board has 7 analog pins that can be used for this purpose. For the full list, see the pinout page (https://adafru.it/AM9) on the main guide.



Trinket M0

A1 is labeled as 2! It's located between "1~" and "3V" on the same side of the board as the little red LED. Ground is located on the opposite side of the board. 3V is located next to 2, on the same end of the board as the reset button.

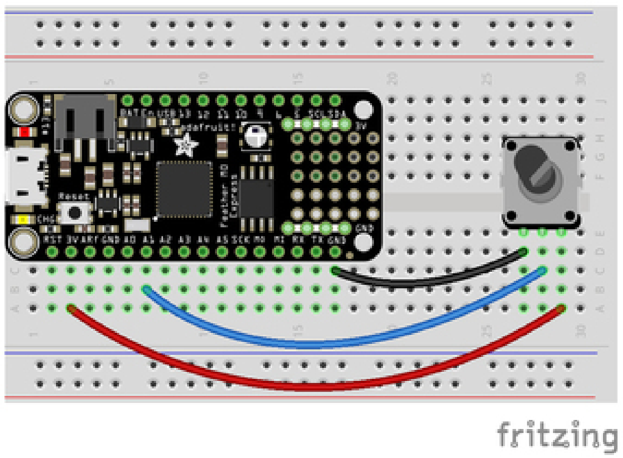
You have 5 analog pins you can use. For the full list, see the pinouts page (https://adafru.it/AMd) on the main guide.



Gemma M0

A1 is located near the top of the board of the board to the left side of the USB Micro port. Ground is on the other side of the USB port from A1. 3V is located to the left side of the battery connector on the bottom of the board.

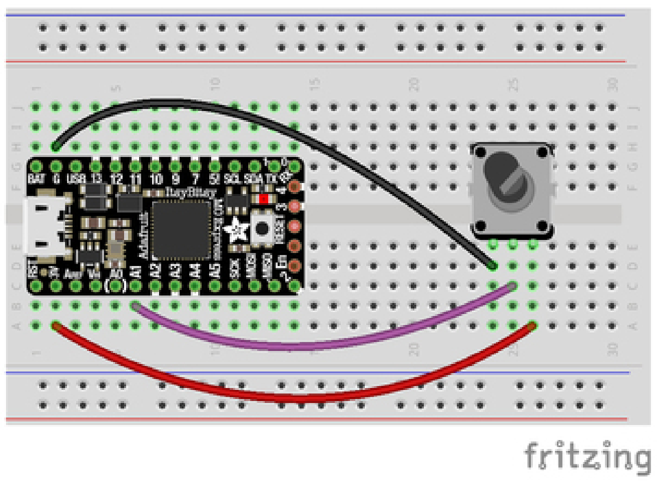
Your board has 3 analog pins. For the full list, see the pinout page (https://adafru.it/AMa) on the main guide.



Feather M0 Express and Feather M4 Express

A1 is located along the edge opposite the battery connector. There are multiple ground pins. 3V is located along the same edge as A1, and is next to the reset button.

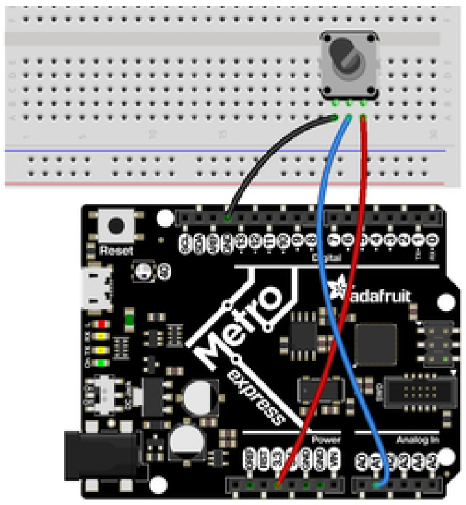
Your board has 6 analog pins you can use. For the full list, see the pinouts page (https://adafru.it/AMc) on the main guide.



ItsyBitsy M0 Express and ItsyBitsy M4 Express

A1 is located in the middle of the board, near the "A" in "Adafruit". Ground is labled "G" and is located next to "BAT", near the USB Micro port. 3V is found on the opposite side of the USB port from Ground, next to RST.

You have 6 analog pins you can use. For a full list, see the pinouts page (https://adafru.it/BMg) on the main guide.



Metro M0 Express and Metro M4 Express

A1 is located on the same side of the board as the barrel jack. There are multiple ground pins available. 3V is labeled "3.3" and is located in the center of the board on the same side as the barrel jack (and as A1).

Your **Metro M0 Express** board has 6 analog pins you can use. For the full list, see the pinouts page (https://adafru.it/AMb) on the main guide.

Your **Metro M4 Express** board has 6 analog pins you can use. For the full list, see the pinouts page (https://adafru.it/B1O) on the main guide.

Reading Analog Pin Values

The `get_voltage()` helper used in the potentiometer example above reads the raw analog pin value and converts it to a voltage level. You can, however, directly read an analog pin value in your code by using `pin.value`. For example, to simply read the raw analog pin value from the potentiometer, you would run the following code:

```
import time
import board
from analogio import AnalogIn

analog_in = AnalogIn(board.A1)

while True:
    print(analog_in.value)
    time.sleep(0.1)
```

This works with any analog pin or input. Use the `<pin_name>.value` to read the raw value and utilise it in your code.

CircuitPython Analog Out

This example shows you how you can set the DAC (true analog output) on pin A0.

A0 is the only true analog output on the M0 boards. No other pins do true analog output!

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython IO demo - analog output
import board
from analogio import AnalogOut

analog_out = AnalogOut(board.A0)

while True:
    # Count up from 0 to 65535, with 64 increment
    # which ends up corresponding to the DAC's 10-bit range
    for i in range(0, 65535, 64):
        analog_out.value = i
```

Creating an analog output

`analog_out = AnalogOut(A0)`

Creates an object `analog_out` and connects the object to **A0**, the only DAC pin available on both the M0 and the M4 boards. (The M4 has two, A0 and A1.)

Setting the analog output

The DAC on the SAMD21 is a 10-bit output, from 0-3.3V. So in theory you will have a resolution of 0.0032 Volts per bit. To allow CircuitPython to be general-purpose enough that it can be used with chips with anything from 8 to 16-bit DACs, the DAC takes a 16-bit value and divides it down internally.

For example, writing 0 will be the same as setting it to 0 - 0 Volts out.

Writing 5000 is the same as setting it to 5000 / 64 = 78, and 78 / 1024 * 3.3V = 0.25V output.

Writing 65535 is the same as 1023 which is the top range and you'll get 3.3V output

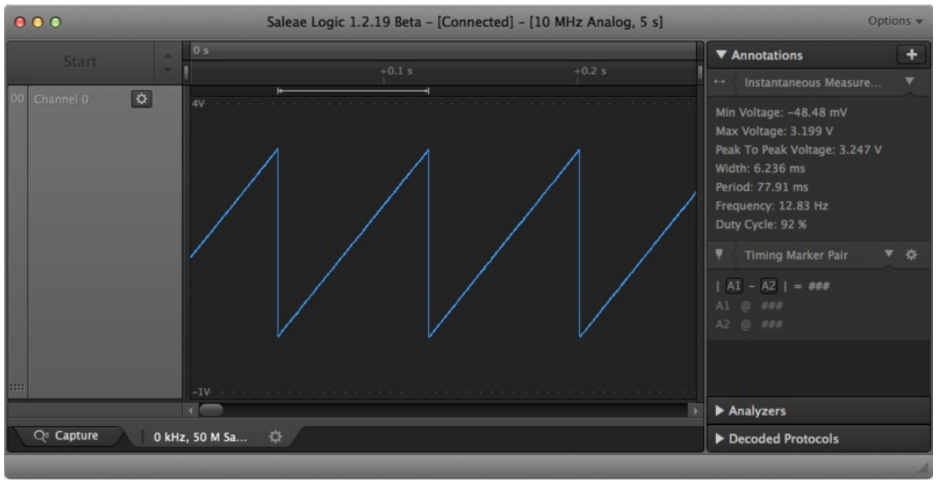
Main Loop

The main loop is fairly simple, it goes through the entire range of the DAC, from 0 to 65535, but increments 64 at a time so it ends up clicking up one bit for each of the 10-bits of range available.

CircuitPython is not terribly fast, so at the fastest update loop you'll get 4 Hz. The DAC isn't good for audio outputs as-is.

Express boards like the Circuit Playground Express, Metro M0 Express, ItsyBitsy M0 Express, ItsyBitsy M4 Express, Metro M4 Express, Feather M4 Express, or Feather M0 Express have more code space and can perform audio playback capabilities via the DAC. Gemma M0 and Trinket M0 cannot!

Check out the Audio Out section of this guide (https://adafru.it/BRj) for examples!



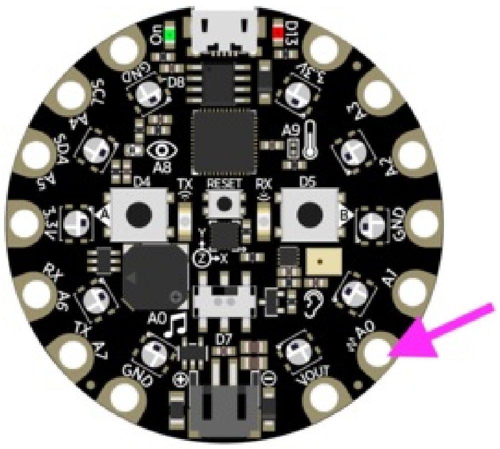
Saleae Logic 1.2.19 Beta – [Connected] – [10 MHz Analog, 5 s]

Min Voltage: -48.48 mV
Max Voltage: 3.199 V
Peak To Peak Voltage: 3.247 V
Width: 6.236 ms
Period: 77.91 ms
Frequency: 12.83 Hz
Duty Cycle: 92 %

| A1 – A2 | = ###
A1 @ ###
A2 @ ###

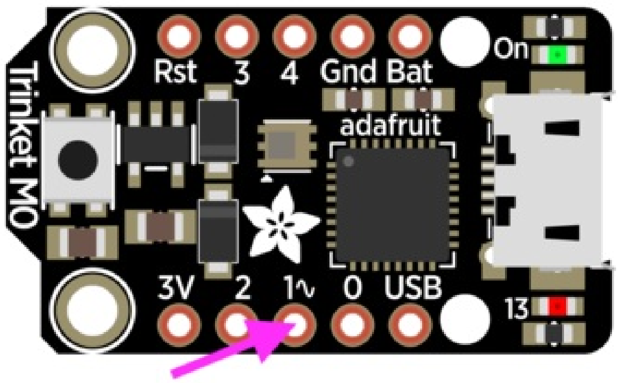
Find the pin

Use the diagrams below to find the A0 pin marked with a magenta arrow!



Circuit Playground Express

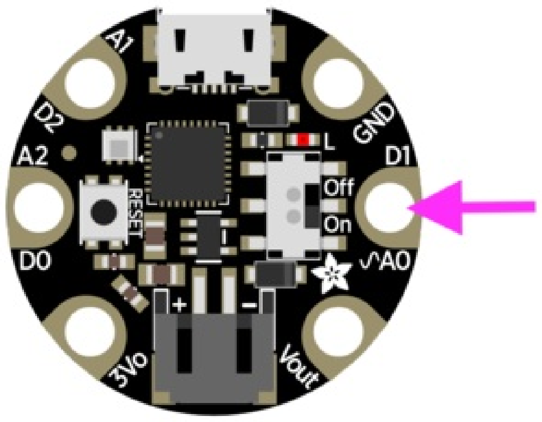
A0 is located between VOUT and A1 near the battery port.



Trinket M0

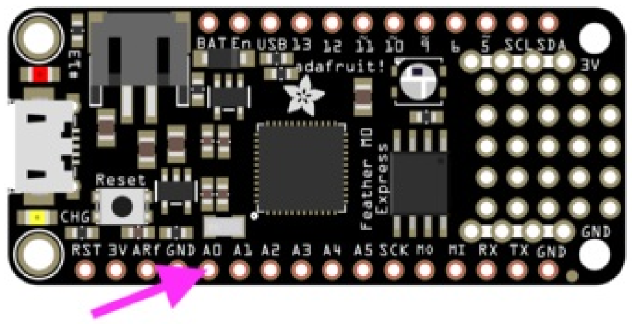
A0 is labeled "1~" on Trinket! A0 is located between "0" and "2" towards the middle of the board on the same side as the red LED.

Gemma M0



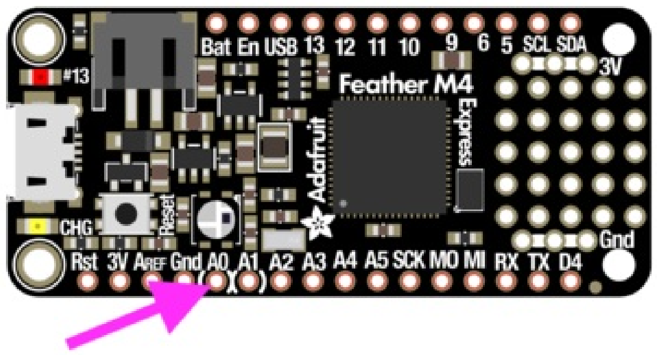
A0 is located in the middle of the right side of the board next to the On/Off switch.

Feather M0 Express


BAT En USB 13 12 11 10 9 6 5 SCL SDA
3V
adafruit!
Feather M0 Express
Reset
CHG
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND
GND

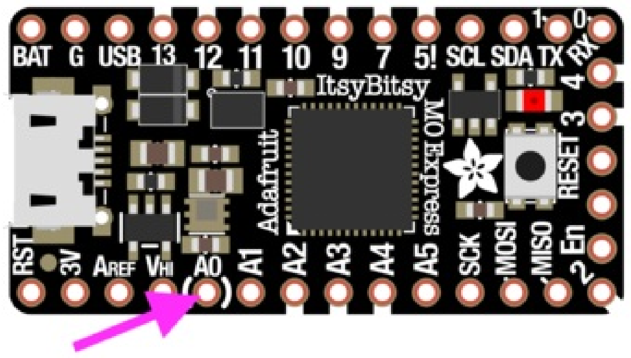
A0 is located between GND and A1 on the opposite side of the board from the battery connector, towards the end with the Reset button.

Feather M4 Express


Bat En USB 13 12 11 10 9 6 5 SCL SDA
3V
#13
Feather M4 Express
Adafruit
Reset
CHG
Rst 3V AREF Gnd A0 A1 A2 A3 A4 A5 SCK MO MI RX TX D4
Gnd

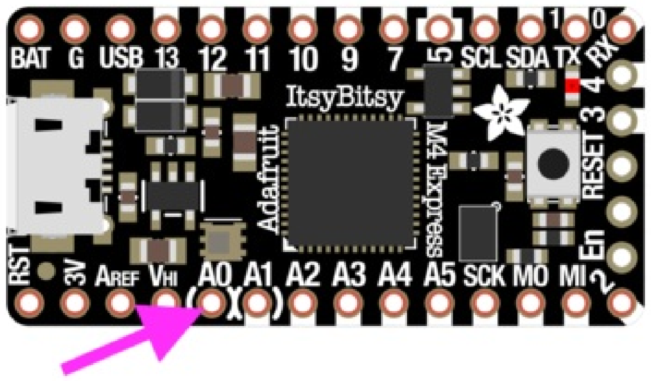
A0 is located between GND and A1 on the opposite side of the board from the battery connector, towards the end with the Reset button, and the pin pad has left and right white parenthesis markings around it

ItsyBitsy M0 Express



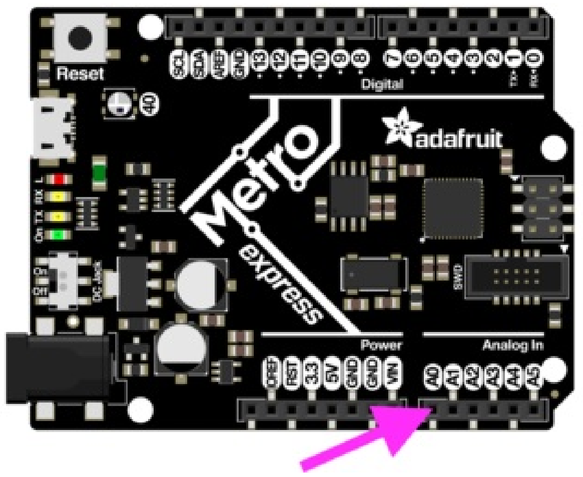
A0 is located between VHI and A1, near the "A" in "Adafruit", and the pin pad has left and right white parenthesis markings around it.

ItsyBitsy M4 Express

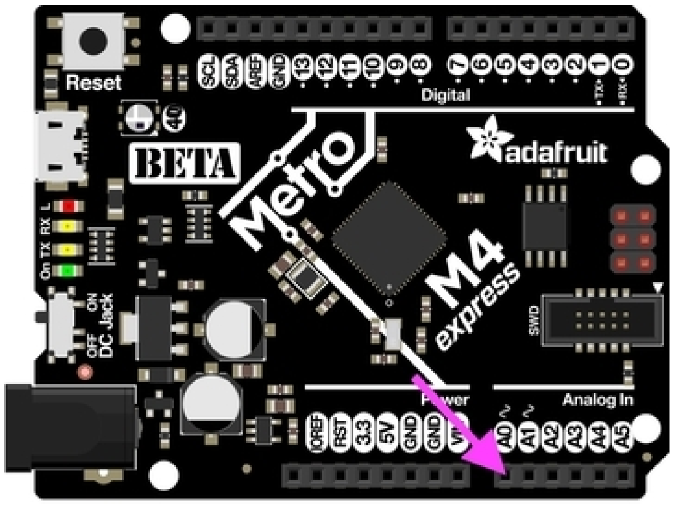


A0 is located between VHI and A1, and the pin pad has left and right white parenthesis markings around it.

Metro M0 Express



A0 is between VIN and A1, and is located along the same side of the board as the barrel jack adapter towards the middle of the headers found on that side of the board.



Metro M4 Express

A0 is between VIN and A1, and is located along the same side of the board as the barrel jack adapter towards the middle of the headers found on that side of the board.

On the Metro M4 Express, there are TWO true analog outputs: A0 and A1.

CircuitPython Audio Out

CircuitPython 3.0 and higher comes with an updated `audioio`, which provides built-in audio output support. You can play generated tones. You can also play, pause and resume wave files. You can have 3V-peak-to-peak analog output or I2S digital output. In this page we will show using analog output.

This is great for all kinds of projects that require sound, like a tone piano or anything where you'd like to add audio effects!

ESP8266, Trinket M0 and Gemma M0 do not support audioio! You must use an M0 Express or M4 Express board for this.

The first example will show you how to generate a tone and play it using a button. The second example will show you how to play, pause, and resume a wave file using a button to resume. Both will play the audio through an audio jack. The default volume on both of these examples is painfully high through headphones. So, we've added a potentiometer and included some code in the tone generation example to control volume.

In our code, we'll use pin A0 for our audio output, as this is the only DAC pin available on every Express board. The M0 Express boards have audio output on A0. The M4 Express boards have two audio output pins, A0 and A1, however we'll be using only A0 in this guide.

Play a Tone

Copy and paste the following code into **code.py** using your favorite editor, and save the file.

```
import time
import array
import math
import audioio
import board
import digitalio

button = digitalio.DigitalInOut(board.A1)
button.switch_to_input(pull=digitalio.Pull.UP)

tone_volume = 0.1  # Increase this to increase the volume of the tone.
frequency = 440  # Set this to the Hz of the tone you want to generate.
length = 8000 // frequency
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int((1 + math.sin(math.pi * 2 * i / length)) * tone_volume * (2 ** 15 - 1))

audio = audioio.AudioOut(board.A0)
sine_wave_sample = audioio.RawSample(sine_wave)

while True:
    if not button.value:
        audio.play(sine_wave_sample, loop=True)
        time.sleep(1)
        audio.stop()
```

First we create the button object, assign it to pin `A1`, and set it as an input with a pull-up. Even though the button switch involves `digitalio`, we're using an A-pin so that the same setup code will work across all the boards.

Since the default volume was incredibly high, we included a `tone_volume` variable in the sine wave code. You can use the code to control the volume by increasing or decreasing this number to increase or decrease the volume. You can also control volume with the potentiometer by rotating the knob.

To set the frequency of the generated tone, change the number assigned to the `frequency` variable to the Hz of the tone you'd like to generate.

Then, we generate one period of a sine wave with the `math.sin` function, and assign it to `sine_wave`.

Next, we create the audio object, and assign it to pin `A0`.

We create a sample of the sine wave by using `RawSample` and providing the `sine_wave` we created.

Inside our loop, we check to see if the button is pressed. The button has two states `True` and `False`. The `button.value` defaults to the `True` state when not pressed. So, to check if it has been pressed, we're looking for the `False` state. So, we check to see `if not button.value` which is the equivalent of `not True`, or `False`.

Once the button is pressed, we `play` the sample we created and we loop it. The `time.sleep(1)` tells it to loop (play) for 1 second. Then we `stop` it after 1 second is up. You can increase or decrease the length of time it plays by increasing or decreasing the number of seconds provided to `time.sleep()`. Try changing it from `1` to `0.5`. Now try changing it to `2`. You can change it to whatever works for you!

That's it!

Play a Wave File

You can use any supported wave file you like. CircuitPython supports **mono or stereo**, at **22 KHz sample rate** (or less) and **16-bit** WAV format. The M0 boards support ONLY MONO. The reason for mono is that there's only one analog output on those boards! The M4 boards support stereo as they have two outputs. The 22 KHz or less because the circuitpython can't handle more data than that (and also it will not sound much better) and the DAC output is 10-bit so anything over 16-bit will just take up room without better quality.

Since the WAV file must fit on the CircuitPython file system, it must be under 2 MB.

CircuitPython does not support OGG or MP3. Just WAV!

We have a detailed guide on how to generate WAV files here (https://adafru.it/s8f).

We've included the one we used here. Download it and copy it to your board.

https://adafru.it/BQF

https://adafru.it/BQF

We're going to play the wave file for 6 seconds, pause it, wait for a button to be pressed, and then resume the file to play through to the end. Then it loops back to the beginning and starts again! Let's take a look.

Copy and paste the following code into **code.py** using your favorite editor, and save the file.

```
import time
import audioio
import board
import digitalio

button = digitalio.DigitalInOut(board.A1)
button.switch_to_input(pull=digitalio.Pull.UP)

wave_file = open("StreetChicken.wav", "rb")
wave = audioio.WaveFile(wave_file)
audio = audioio.AudioOut(board.A0)

while True:
    audio.play(wave)

    # This allows you to do other things while the audio plays!
    t = time.monotonic()
    while time.monotonic() - t < 6:
        pass

    audio.pause()
    print("Waiting for button press to continue!")
    while button.value:
        pass
    audio.resume()
    while audio.playing:
        pass
    print("Done!")
```

First we create the button object, assign it to pin `A1`, and set it as an input with a pull-up.

Next we then open the file, `"StreetChicken.wav"` as a **r**eadable **b**inary and store the file object in `wave_file` which is what we use to actually read audio from: `wave_file = open("StreetChicken.wav", "rb")`.

Now we will ask the audio playback system to load the wave data from the file `wave = audioio.WaveFile(wave_file)` and finally request that the audio is played through the A0 analog output pin `audio = audioio.AudioOut(board.A0)`.

The audio file is now ready to go, and can be played at any time with `audio.play(wave)`!

Inside our loop, we start by playing the file.

Next we have the block that tells the code to wait 6 seconds before pausing the file. We chose to go with using `time.monotonic()` because it's **non-blocking** which means you can do other things while the file is playing, like control servos or NeoPixels! At any given point in time, `time.monotonic()` is equal to the number seconds since your board was last power-cycled. (The soft-reboot that occurs with the auto-reload when you save changes to your CircuitPython code, or enter and exit the REPL, does not start it over.) When it is called, it returns a number with a decimal. When you assign `time.monotonic()` to a variable, that variable is equal to the number of seconds that `time.monotonic()` was equal to at the moment the variable was assigned. You can then call it again and subtract the variable from `time.monotonic()` to get the amount of time that has passed. For more details, check out this example (https://adafru.it/BlT).

So, we assign `t = time.monotonic()` to get a starting point. Then we say `pass`, or "do nothing" until the difference between `t` and `time.monotonic()` is greater than `6` seconds. In other words, continue playing until 6 seconds passes. Remember, you can add in other code here to do other things while you're playing audio for 6 seconds.

Then we `pause` the audio and `print` to the serial console, `"Waiting for button press to continue!"`

Now we're going to wait for a button press in the same way we did for playing the generated tone. We're saying `while button.value`, or while the button is returning `True`, `pass`. Once the button is pressed, it returns `False`, and this tells the code to continue.

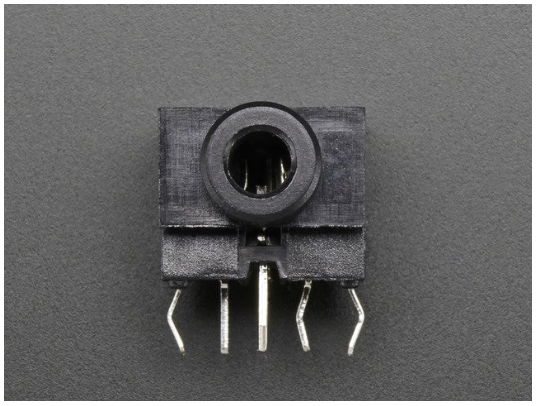
Once the button is pressed, we `resume` playing the file. We tell it to finish playing saying `while audio.playing: pass`.

Finally, we `print` to the serial console, `"Done!"`

You can do this with any supported wave file, and you can include all kinds of things in your project while the file is playing. Give it a try!

Wire It Up

Along with your **microcontroller board**, we're going to be using:



Breadboard-Friendly 3.5mm Stereo Headphone Jack

OUT OF STOCK

Out Of Stock

Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack

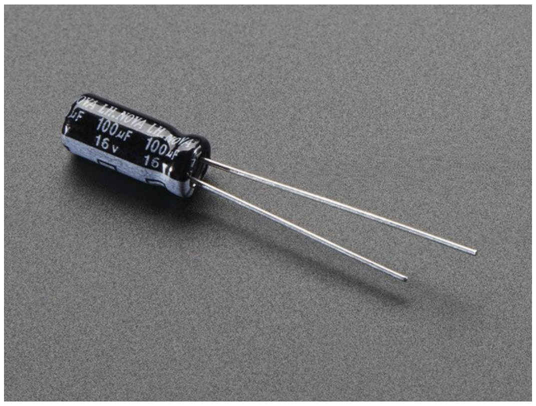
$2.50
IN STOCK

Add To Cart

Panel Mount 10K potentiometer (Breadboard Friendly)

OUT OF STOCK

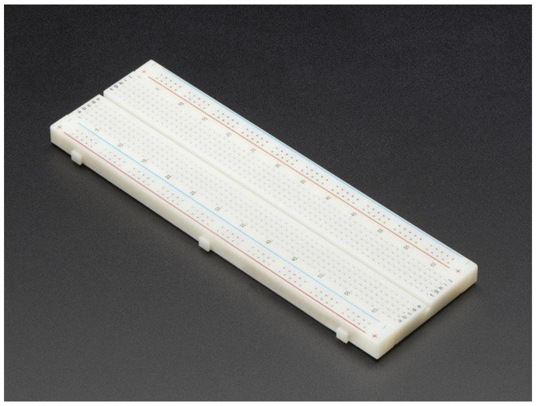
Out Of Stock



100uF 16V Electrolytic Capacitors - Pack of 10

$1.95
IN STOCK

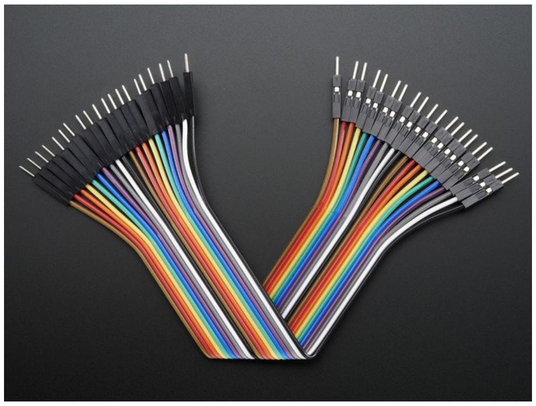
Add To Cart



Full sized breadboard

$5.95
IN STOCK

Add To Cart

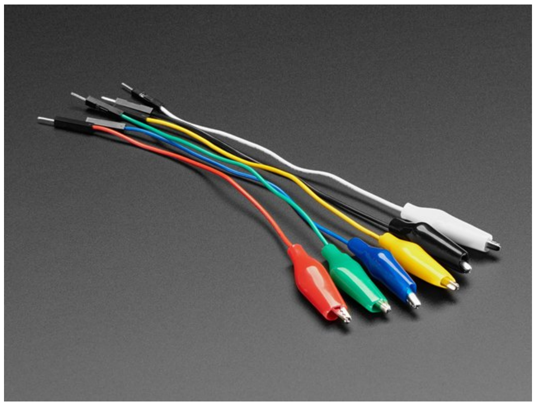


Premium Male/Male Jumper Wires - 20 x 6" (150mm)

$1.95
IN STOCK

Add To Cart

And to make it easier to wire up the Circuit Playground Express:



Small Alligator Clip to Male Jumper Wire Bundle - 6 Pieces

$3.95
IN STOCK

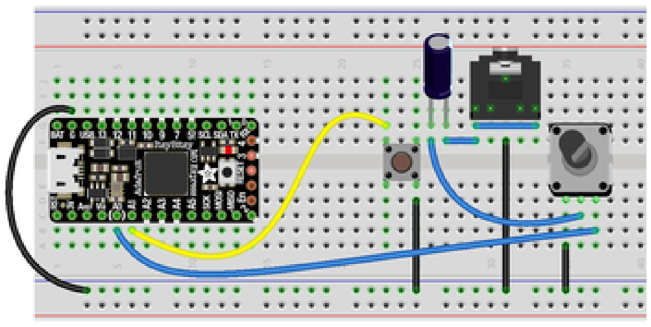
Add To Cart

Button switches with four pins are really two pairs of pins. When wiring up a button switch with four pins, the easiest way to verify that you're wiring up the correct pins is to **wire up opposite corners of the button switch**. Then there's no chance that you'll accidentally wire up the same pin twice.

Here are the steps you're going to follow to wire up these components:

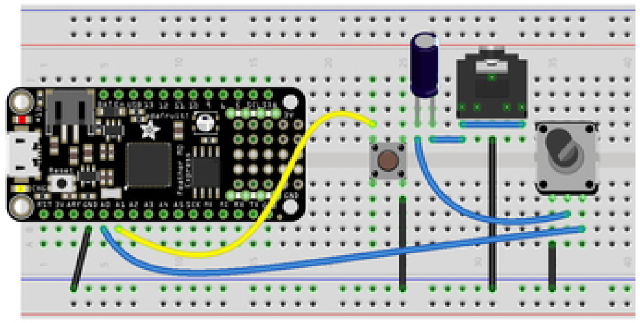
- Connect the **ground pin on your board** to a **ground rail on the breadboard** because you'll be connecting all three components to ground.
- Connect **one pin on the button switch** to **pin A1 on your board**, and the **opposite pin on the button switch** to the **ground rail on the breadboard**.
- Connect the **left and right pin on the audio jack to each other**.
- Connect the **center pin on the audio jack** to the **ground rail on the breadboard**.
- Connect the **left pin** to the **negative side of a 100mF capacitor.**
- Connect the **positive side of the capacitor** to the **center pin on the potentiometer**.
- Connect the **right pin on the potentiometer** to **pin A0 on your board**.
- Connect the **left pin of the potentiometer** to **the ground rail on the breadboard.**

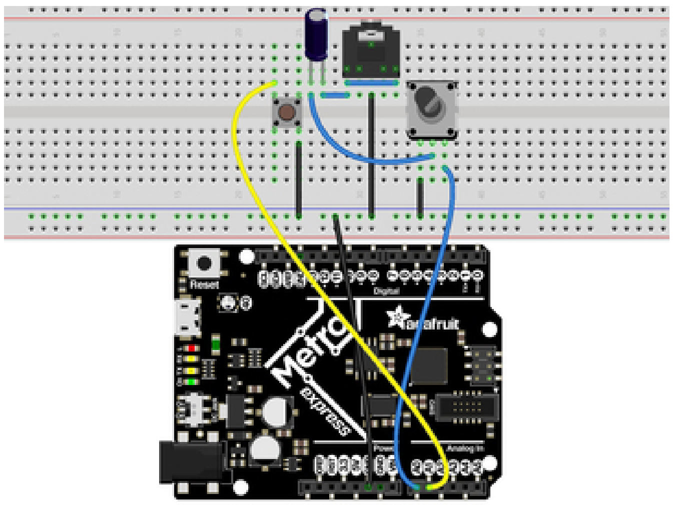
The list below shows wiring diagrams to help with finding the correct pins and wiring up the different components. The ground wires are black. The wire for the button switch is yellow. The wires involved with audio are blue.

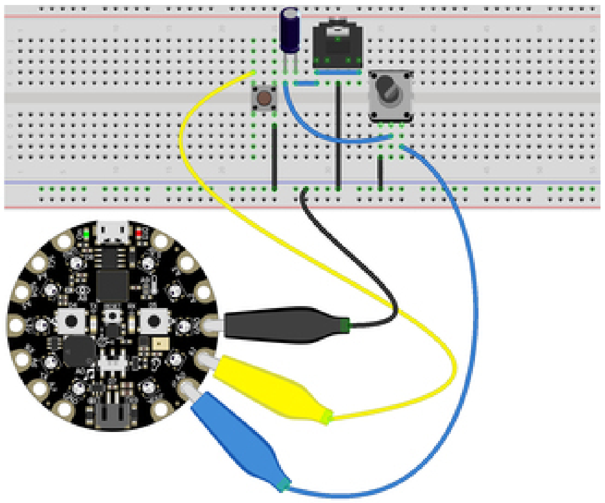


Wiring is the same for the M4 versions of the boards as it is for the M0 versions. Follow the same image for both.

Use a breadboard to make your wiring neat and tidy!







Circuit Playground Express is wired electrically the same as the ItsyBitsy/Feather/Metro above but we use alligator clip to jumper wires instead of plain jumpers

Your board has pulseio support, which means you can PWM LEDs, control servos, beep piezos, and manage "pulse train" type devices like DHT22 and Infrared.

Nearly every pin has PWM support! For example, all ATSAMD21 board have an **A0** pin which is 'true' analog out and *does not* have PWM support.

PWM with Fixed Frequency

This example will show you how to use PWM to fade the little red LED on your board.

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
import time
import board
import pulseio

led = pulseio.PWMOut(board.D13, frequency=5000, duty_cycle=0)

while True:
    for i in range(100):
        # PWM LED up and down
        if i < 50:
            led.duty_cycle = int(i * 2 * 65535 / 100)  # Up
        else:
            led.duty_cycle = 65535 - int((i - 50) * 2 * 65535 / 100)  # Down
        time.sleep(0.01)
```

Create a PWM Output

`led = pulseio.PWMOut(board.D13, frequency=5000, duty_cycle=0)`

Since we're using the onboard LED, we'll call the object led, use pulseio.PWMOut to create the output and pass in the D13 LED pin to use.

Main Loop

The main loop uses range() to cycle through the loop. When the range is below 50, it PWMs the LED brightness up, and when the range is above 50, it PWMs the brightness down. This is how it fades the LED brighter and dimmer!

The time.sleep() is needed to allow the PWM process to occur over a period of time. Otherwise it happens too quickly for you to see!

PWM Output with Variable Frequency

Fixed frequency outputs are great for pulsing LEDs or controlling servos. But if you want to make some beeps with a piezo, you'll need to vary the frequency.

The following example uses pulseio to make a series of tones on a piezo.

To use with any of the M0 boards, no changes to the following code are needed.

To use with the Metro M4 Express, ItsyBitsy M4 Express or the Feather M4 Express, you must comment out the `piezo = pulseio.PWMOut(board.A2, duty_cycle=0, frequency=440, variable_frequency=True)` line and uncomment the `piezo = pulseio.PWMOut(board.A1, duty_cycle=0, frequency=440, variable_frequency=True)` line. A2 is not a supported PWM pin on the M4 boards!

Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

```
import time
import board
import pulseio

# For the M0 boards:
piezo = pulseio.PWMOut(board.A2, duty_cycle=0, frequency=440, variable_frequency=True)

# For the M4 boards:
# piezo = pulseio.PWMOut(board.A1, duty_cycle=0, frequency=440, variable_frequency=True)

while True:
    for f in (262, 294, 330, 349, 392, 440, 494, 523):
        piezo.frequency = f
        piezo.duty_cycle = 65536 // 2  # On 50%
        time.sleep(0.25)  # On for 1/4 second
        piezo.duty_cycle = 0  # Off
        time.sleep(0.05)  # Pause between notes
    time.sleep(0.5)
```

If you have `simpleio` library loaded into your /lib folder on your board, we have a nice little helper that makes a tone for you on a piezo with a single command.

To use with any of the M0 boards, no changes to the following code are needed.

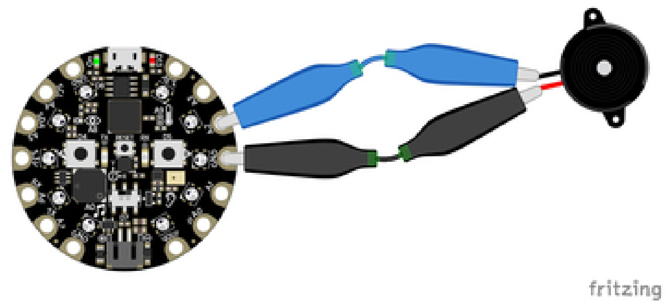
To use with the Metro M4 Express, ItsyBitsy M4 Express or the Feather M4 Express, you must comment out the `simpleio.tone(board.A2, f, 0.25)` line and uncomment the `simpleio.tone(board.A1, f, 0.25)` line. A2 is not a supported PWM pin on the M4 boards!

```
import time
import board
import simpleio

while True:
    for f in (262, 294, 330, 349, 392, 440, 494, 523):
        # For the M0 boards:
        simpleio.tone(board.A2, f, 0.25)  # on for 1/4 second
        # For the M4 boards:
        # simpleio.tone(board.A1, f, 0.25)  # on for 1/4 second
        time.sleep(0.05)  # pause between notes
    time.sleep(0.5)
```

As you can see, it's much simpler!

Wire it up

Use the diagrams below to help you wire up your piezo. Attach one leg of the piezo to pin **A2** on the M0 boards or **A1** on the M4 boards, and the other leg to **ground**. It doesn't matter which leg is connected to which pin. They're interchangeable!

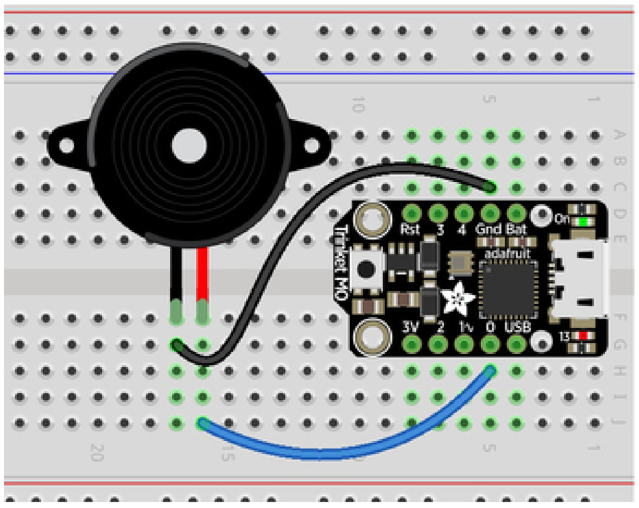


Circuit Playground Express

Use alligator clips to attach **A2** and any one of the **GND** to different legs of the piezo.

CPX has PWM on the following pins: A1, A2, A3, A6, RX, LIGHT, A8, TEMPERATURE, A9, BUTTON_B, D5, SLIDE_SWITCH, D7, D13, REMOTEIN, IR_RX, REMOTEOUT, IR_TX, IR_PROXIMITY, MICROPHONE_CLOCK, MICROPHONE_DATA, ACCELEROMETER_INTERRUPT, ACCELEROMETER_SDA, ACCELEROMETER_SCL, SPEAKER_ENABLE.

There is NO PWM on: A0, SPEAKER, A4, SCL, A5, SDA, A7, TX, BUTTON_A, D4, NEOPIXEL, D8, SCK, MOSI, MISO, FLASH_CS.



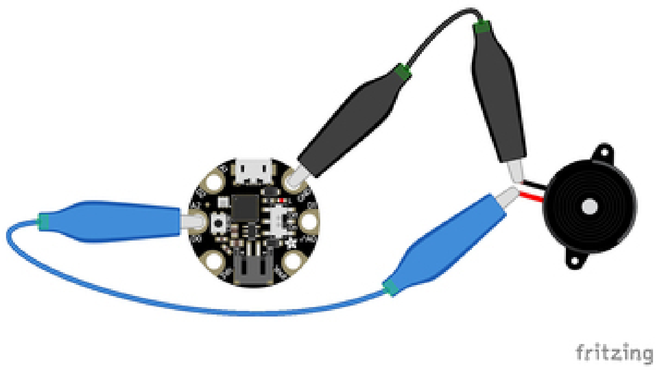
Trinket M0

Note: A2 on Trinket is also labeled Digital "0"!

Use jumper wires to connect **GND** and **D0** to different legs of the piezo.

Trinket has PWM available on the following pins: D0, A2, SDA, D2, A1, SCL, MISO, D4, A4, TX, MOSI, D3, A3, RX, SCK, D13, APA102_MOSI, APA102_SCK.

There is NO PWM on: A0, D1.

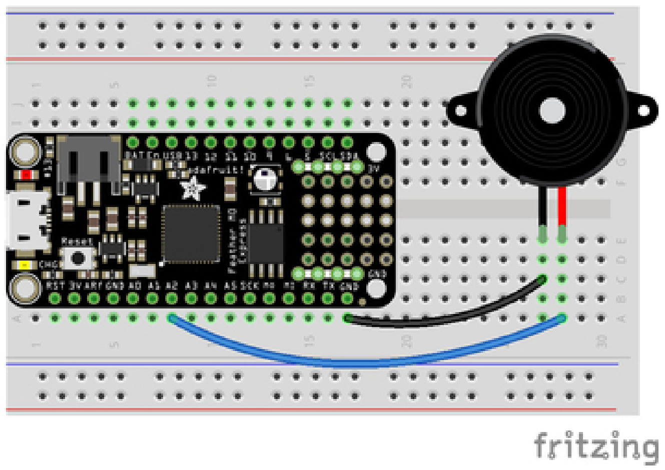


Gemma M0

Use alligator clips to attach **A2** and **GND** to different legs on the piezo.

Gemma has PWM available on the following pins: A1, D2, RX, SCL, A2, D0, TX, SDA, L, D13, APA102_MOSI, APA102_SCK.

There is NO PWM on: A0, D1.

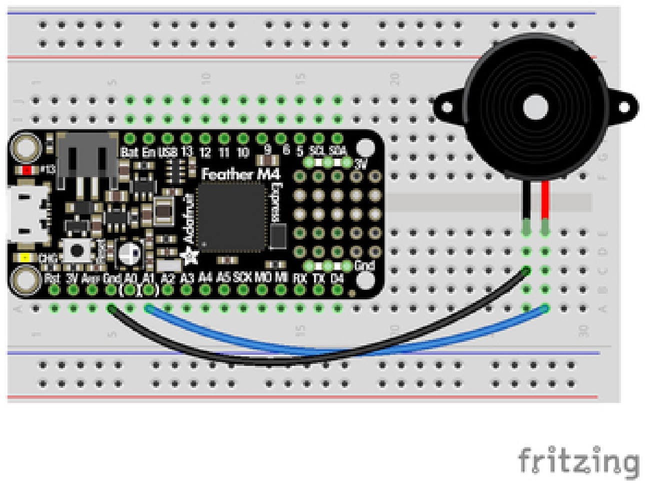


Feather M0 Express

Use jumper wires to attach **A2** and one of the two **GND** to different legs of the piezo.

Feather M0 Express has PWM on the following pins: A2, A3, A4, SCK, MOSI, MISO, D0, RX, D1, TX, SDA, SCL, D5, D6, D9, D10, D11, D12, D13, NEOPIXEL.

There is NO PWM on: A0, A1, A5.



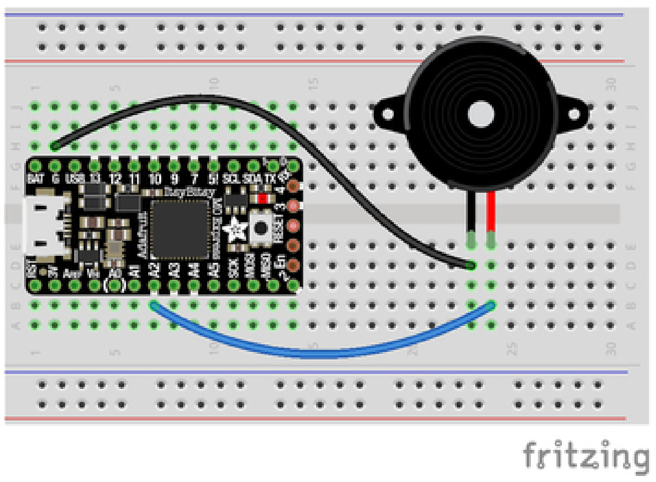
Feather M4 Express

Use jumper wires to attach **A1** and one of the two **GND** to different legs of the piezo.

To use A1, comment out the current pin setup line, and uncomment the line labeled for the M4 boards. See the details above!

Feather M4 Express has PWM on the following pins: A1, A3, SCK, D0, RX, D1, TX, SDA, SCL, D4, D5, D6, D9, D10, D11, D12, D13.

There is NO PWM on: A0, A2, A4, A5, MOSI, MISO.

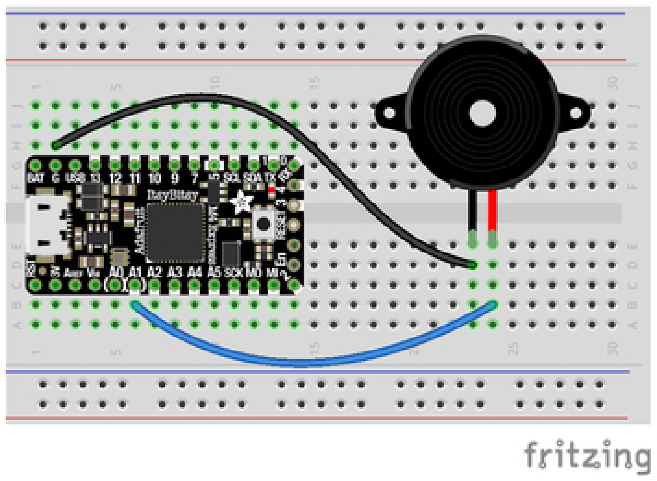


ItsyBitsy M0 Express

Use jumper wires to attach **A2** and **G** to different legs of the piezo.

ItsyBitsy M0 Express has PWM on the following pins: D0, RX, D1, TX, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, L, A2, A3, A4, MOSI, MISO, SCK, SCL, SDA, APA102_MOSI, APA102_SCK.

There is NO PWM on: A0, A1, A5.



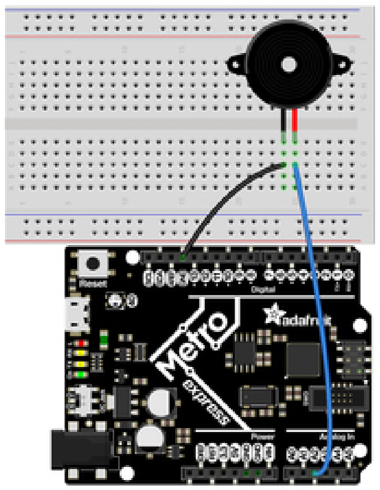
ItsyBitsy M4 Express

Use jumper wires to attach **A1** and **G** to different legs of the piezo.

To use A1, comment out the current pin setup line, and uncomment the line labeled for the M4 boards. See the details above!

ItsyBitsy M4 Express has PWM on the following pins: A1, D0, RX, D1, TX, D2, D4, D5, D7, D9, D10, D11, D12, D13, SDA, SCL.

There is NO PWM on: A2, A3, A4, A5, D3, SCK, MOSI, MISO.

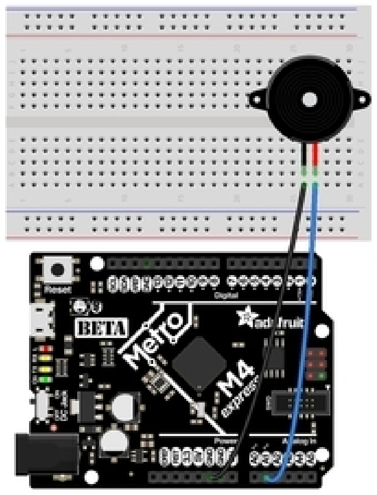


Metro M0 Express

Use jumper wires to connect **A2** and any one of the **GND** to different legs on the piezo.

Metro M0 Express has PWM on the following pins: A2, A3, A4, D0, RX, D1, TX, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, SDA, SCL, NEOPIXEL, SCK, MOSI, MISO.

There is NO PWM on: A0, A1, A5, FLASH_CS.



Metro M4 Express

Use jumper wires to connect **A1** and any one of the **GND** to different legs on the piezo.

To use A1, comment out the current pin setup line, and uncomment the line labeled for the M4 boards. See the details above!

Metro M4 Express has PWM on: A1, A5, D0, RX, D1, TX, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, SDA, SCK, MOSI, MISO

There is No PWM on: A0, A2, A3, A4, SCL, AREF, NEOPIXEL, LED_RX, LED_TX.

Where's My PWM?

Want to check to see which pins have PWM yourself? We've written this handy script! It attempts to setup PWM on every pin available, and lets you know which ones work and which ones don't. Check it out!

```
import board
import pulseio

for pin_name in dir(board):
    pin = getattr(board, pin_name)
    try:
        p = pulseio.PWMOut(pin)
        p.deinit()
        print("PWM on:", pin_name)  # Prints the valid, PWM-capable pins!
    except ValueError:  # This is the error returned when the pin is invalid.
        print("No PWM on:", pin_name)  # Prints the invalid pins.
    except RuntimeError:  # Timer conflict error.
        print("Timers in use:", pin_name)  # Prints the timer conflict pins.
    except TypeError:  # Error returned when checking a non-pin object in dir(board).
        pass  # Passes over non-pin objects in dir(board).
```

CircuitPython Servo

In order to use servos, we take advantage of `pulseio`. Now, in theory, you could just use the raw `pulseio` calls to set the frequency to 50 Hz and then set the pulse widths. But we would rather make it a little more elegant and easy!

So, instead we will use `adafruit_motor` which manages servos for you quite nicely! `adafruit_motor` is a library so be sure to grab it from the library bundle if you have not yet (https://adafru.it/zdx)! If you need help installing the library, check out the CircuitPython Libraries page (https://adafru.it/ABU).

Servos come in two types:

- A **standard hobby servo** - the horn moves 180 degrees (90 degrees in each direction from zero degrees).
- A **continuous servo** - the horn moves in full rotation like a DC motor. Instead of an angle specified, you set a throttle value with 1.0 being full forward, 0.5 being half forward, 0 being stopped, and -1 being full reverse, with other values between.

Servo Wiring

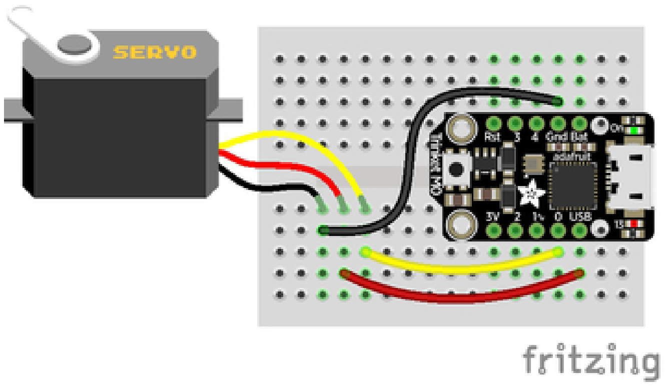
Servos will only work on PWM-capable pins! Check your board details to verify which pins have PWM outputs.

The connections for a servo are the same for standard servos and continuous rotation servos.

Connect the servo's **brown** or **black** ground wire to ground on the CircuitPython board.

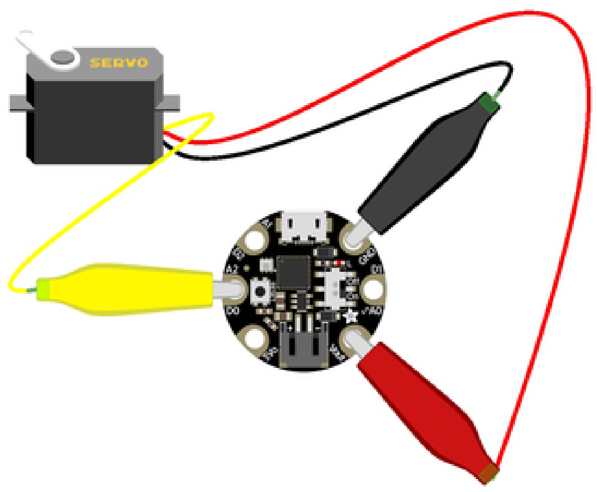
Connect the servo's **red** power wire to 5V power, USB power is good for a servo or two. For more than that, you'll need an external battery pack. Do not use 3.3V for powering a servo!

Connect the servo's **yellow** or **white** signal wire to the control/data pin, in this case **A1 or A2** but you can use any PWM-capable pin.

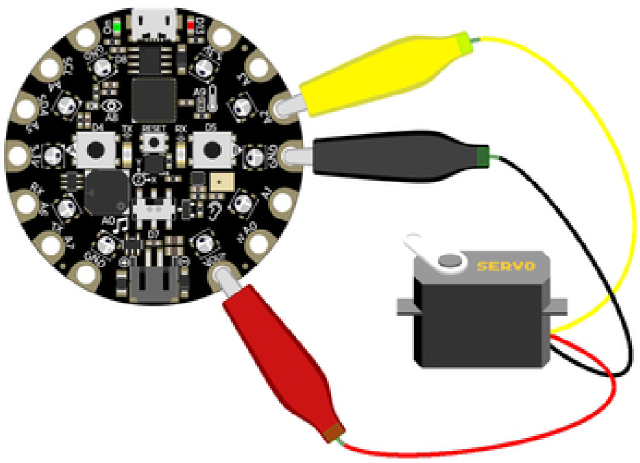


For example, to wire a servo to **Trinket**, connect the ground wire to **GND**, the power wire to **USB**, and the signal wire to **0**.

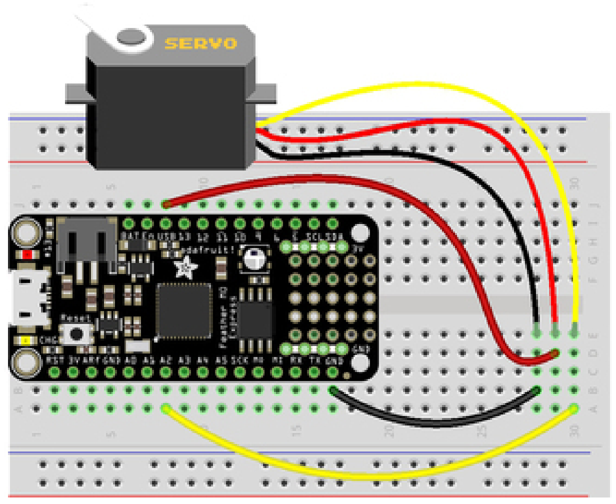
Remember, **A2 on Trinket is labeled "0"**.



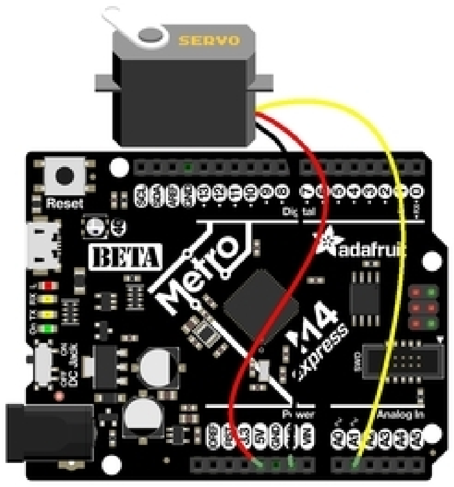
For **Gemma**, use jumper wire alligator clips to connect the ground wire to **GND**, the power wire to **VOUT**, and the signal wire to **A2**.



For **Circuit Playground Express and Circuit Playground Bluefruit**, use jumper wire alligator clips to connect the ground wire to **GND**, the power wire to **VOUT**, and the signal wire to **A2**.



For boards like **Feather M0 Express**, **ItsyBitsy M0 Express** and **Metro M0 Express**, connect the ground wire to any **GND**, the power wire to **USB or 5V**, and the signal wire to **A2**.



For the **Metro M4 Express**, **ItsyBitsy M4 Express** and the **Feather M4 Express**, connect the ground wire to any **G or GND**, the power wire to **USB or 5V**, and the signal wire to **A1**.

Standard Servo Code

Here's an example that will sweep a servo connected to pin **A2** from 0 degrees to 180 degrees (-90 to 90 degrees) and back:

```
import time
import board
import pulseio
from adafruit_motor import servo

# create a PWMOut object on Pin A2.
pwm = pulseio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.Servo(pwm)

while True:
    for angle in range(0, 180, 5):  # 0 - 180 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
    for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
```

Continuous Servo Code

There are two differences with Continuous Servos vs. Standard Servos:

1. The `servo` object is created like `my_servo = servo.ContinuousServo(pwm)` instead of `my_servo = servo.Servo(pwm)`
2. Instead of using `myservo.angle`, you use `my_servo.throttle` using a throttle value from 1.0 (full on) to 0.0 (stopped) to -1.0 (full reverse). Any number between would be a partial speed forward (positive) or reverse (negative). This is very similar to standard DC motor control with the **adafruit_motor** library.

This example runs full forward for 2 seconds, stops for 2 seconds, runs full reverse for 2 seconds, then stops for 4 seconds.

```
# Continuous Servo Test Program for CircuitPython
import time
import board
import pulseio
from adafruit_motor import servo

# create a PWMOut object on Pin A2.
pwm = pulseio.PWMOut(board.A2, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.ContinuousServo(pwm)

while True:
    print("forward")
    my_servo.throttle = 1.0
    time.sleep(2.0)
    print("stop")
    my_servo.throttle = 0.0
    time.sleep(2.0)
    print("reverse")
    my_servo.throttle = -1.0
    time.sleep(2.0)
    print("stop")
    my_servo.throttle = 0.0
    time.sleep(4.0)
```

Pretty simple!

Note that we assume that 0 degrees is 0.5ms and 180 degrees is a pulse width of 2.5ms. That's a bit wider than the *official* 1-2ms pulse widths. If you have a servo that has a different range you can initialize the `servo` object with a different `min_pulse` and `max_pulse`. For example:

`my_servo = servo.Servo(pwm, min_pulse = 500, max_pulse = 2500)`

For more detailed information on using servos with CircuitPython, check out the CircuitPython section of the servo guide (https://adafru.it/Bei)!

CircuitPython Cap Touch

Every CircuitPython designed M0 board has capacitive touch capabilities. This means each board has at least one pin that works as an input when you touch it! The capacitive touch is done *completely* in hardware, so no external resistors, capacitors or ICs required. Which is really nice!

Capacitive touch is not supported on the M4 Express boards.

This example will show you how to use a capacitive touch pin on your board.

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
import time

import board
import touchio

touch_pad = board.A0  # Will not work for Circuit Playground Express!
# touch_pad = board.A1  # For Circuit Playground Express

touch = touchio.TouchIn(touch_pad)

while True:
    if touch.value:
        print("Touched!")
    time.sleep(0.05)
```

Create the Touch Input

First, we assign the variable `touch_pad` to a pin. The example uses **A0**, so we assign `touch_pad = board.A0`. You can choose any touch capable pin from the list below if you'd like to use a different pin. Then we create the touch object, name it `touch` and attach it to `touch_pad`.

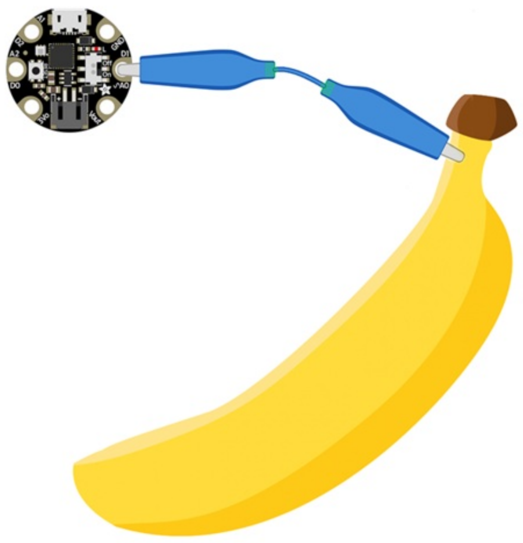
To use with Circuit Playground Express, comment out `touch_pad = board.A0` and uncomment `touch_pad = board.A1`.

Main Loop

Next, we create a loop that checks to see if the pin is touched. If it is, it `prints` to the serial console. Connect to the serial console to see the printed results when you touch the pin!

Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

No extra hardware is required, because you can touch the pin directly. However, you may want to attach alligator clips or copper tape to metallic or conductive objects. Try metal flatware, fruit or other foods, liquids, aluminum foil, or other items lying around your desk!



You may need to reload your code or restart your board after changing the attached item because the capacitive touch code "calibrates" based on what it sees when it first starts up. So if you get too many touch responses or not enough, reload your code through the serial console or eject the board and tap the reset button!

Find the Pin(s)

Your board may have more touch capable pins beyond A0. We've included a list below that helps you find A0 (or A1 in the case of CPX) for this example, identified by the magenta arrow. This list also includes information about any other pins that work for touch on each board!

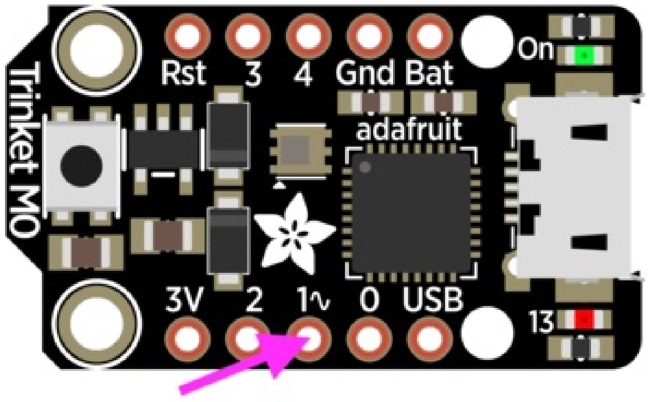
To use the other pins, simply change the number in **A0** to the pin you want to use. For example, if you want to use **A3** instead, your code would start with `touch_pad = board.A3` .

If you would like to use more than one pin at the same time, your code may look like the following. If needed, you can modify this code to include pins that work for your board.

```
# CircuitPython Demo - Cap Touch Multiple Pins
# Example does NOT work with Trinket M0!

import time

import board
import touchio

touch_A1 = touchio.TouchIn(board.A1)  # Not a touch pin on Trinket M0!
touch_A2 = touchio.TouchIn(board.A2)  # Not a touch pin on Trinket M0!

while True:
    if touch_A1.value:
        print("Touched A1!")
    if touch_A2.value:
        print("Touched A2!")
    time.sleep(0.05)
```

This example does NOT work for Trinket M0! You must change the pins to use with this board. This example only works with Gemma, Circuit Playground Express, Feather M0 Express, Metro M0 Express and ItsyBitsy M0 Express.

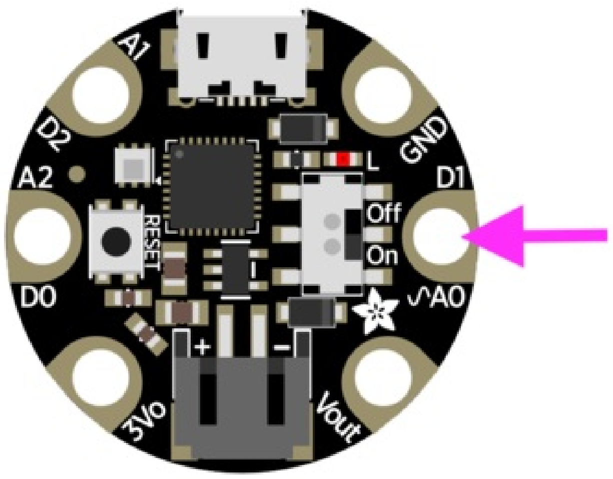
Use the list below to find out what pins you can use with your board. Then, try adding them to your code and have fun!



Trinket M0

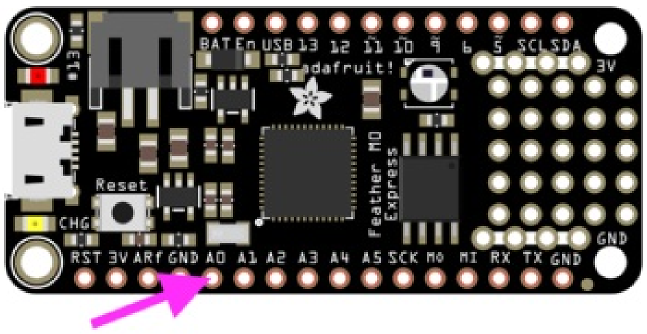
There are three touch capable pins on Trinket: **A0**, **A3**, and **A4**.

Remember, **A0 is labeled "1~" on Trinket M0!**



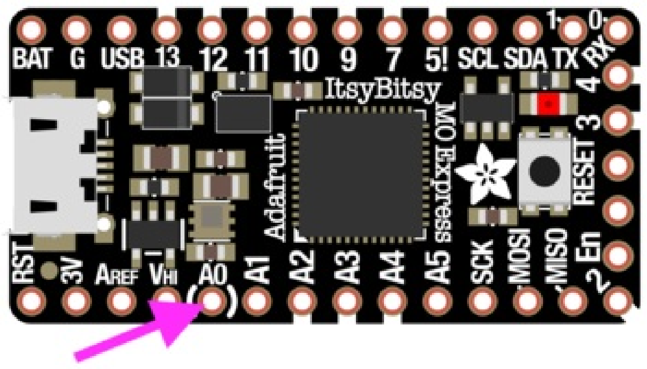
Gemma M0

There are three pins on Gemma, in the form of alligator-clip-friendly pads, that work for touch input: **A0**, **A1** and **A2**.



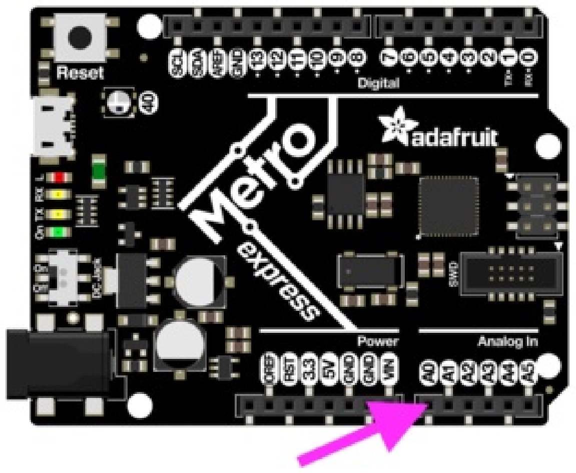
Feather M0 Express

There are 6 pins on the Feather that have touch capability: **A0 - A5**.


BAT G USB 13 12 11 10 9 7 5! SCL SDA TX RX
ItsyBitsy M0 Express
Adafruit
RST 3V AREF VHI A0 A1 A2 A3 A4 A5 SCK MOSI MISO En
RESET

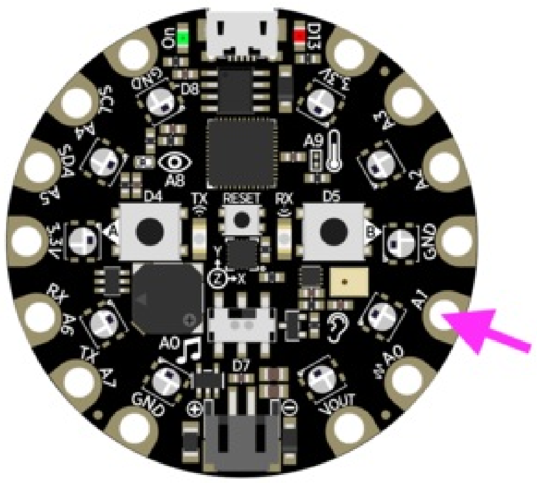
ItsyBitsy M0 Express

There are 6 pins on the ItsyBitsy that have touch capability: **A0 - A5**.



Metro M0 Express

There are 6 pins on the Metro that have touch capability: **A0 - A5**.



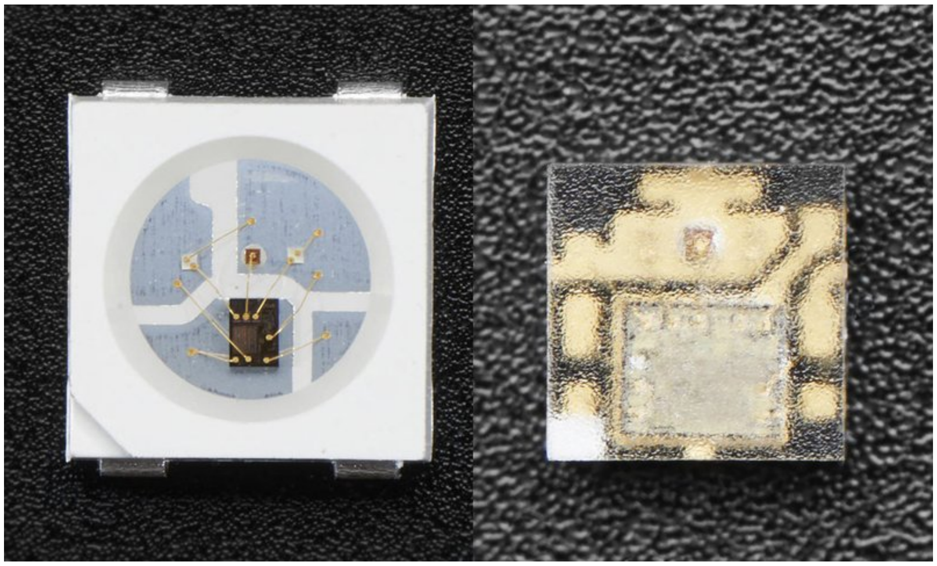
Circuit Playground Express

Circuit Playground Express has seven touch capable pins! You have **A1 - A7** available, in the form of alligator-clip-friendly pads. See the CPX guide Cap Touch section (https://adafru.it/ANC) for more information on using these pads for touch!

Remember: A0 does **NOT** have touch capabilities on CPX.

CircuitPython Internal RGB LED

Every board has a built in RGB LED. You can use CircuitPython to control the color and brightness of this LED. There are two different types of internal RGB LEDs: DotStar (https://adafru.it/kDg) and NeoPixel (https://adafru.it/Bej). This section covers both and explains which boards have which LED.



The first example will show you how to change the color and brightness of the internal RGB LED.

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
import time
import board

# For Trinket M0, Gemma M0, ItsyBitsy M0 Express, and ItsyBitsy M4 Express
import adafruit_dotstar
led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
# For Feather M0 Express, Metro M0 Express, Metro M4 Express, and Circuit Playground Express
# import neopixel
# led = neopixel.NeoPixel(board.NEOPIXEL, 1)

led.brightness = 0.3

while True:
    led[0] = (255, 0, 0)
    time.sleep(0.5)
    led[0] = (0, 255, 0)
    time.sleep(0.5)
    led[0] = (0, 0, 255)
    time.sleep(0.5)
```

Create the LED

First, we create the LED object and attach it to the correct pin or pins. In the case of a NeoPixel, there is only one pin necessary, and we have called it `NEOPIXEL` for easier use. In the case of a DotStar, however, there are two pins necessary, and so we use the pin names `APA102_MOSI` and `APA102_SCK` to get it set up. Since we're using the single onboard LED, the last thing we do is tell it that there's only `1` LED!

Trinket M0, **Gemma M0**, **ItsyBitsy M0 Express**, and **ItsyBitsy M4 Express** each have an onboard **Dotstar** LED, so **no changes are needed** to the initial version of the example.

Feather M0 Express, Feather M4 Express, Metro M0 Express, Metro M4 Express, and Circuit Playground Express each have an onboard **NeoPixel** LED, so you must **comment out** `import adafruit_dotstar` **and** `led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)`, and **uncomment** `import neopixel` **and** `led = neopixel.NeoPixel(board.NEOPIXEL, 1)`.

Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

Brightness

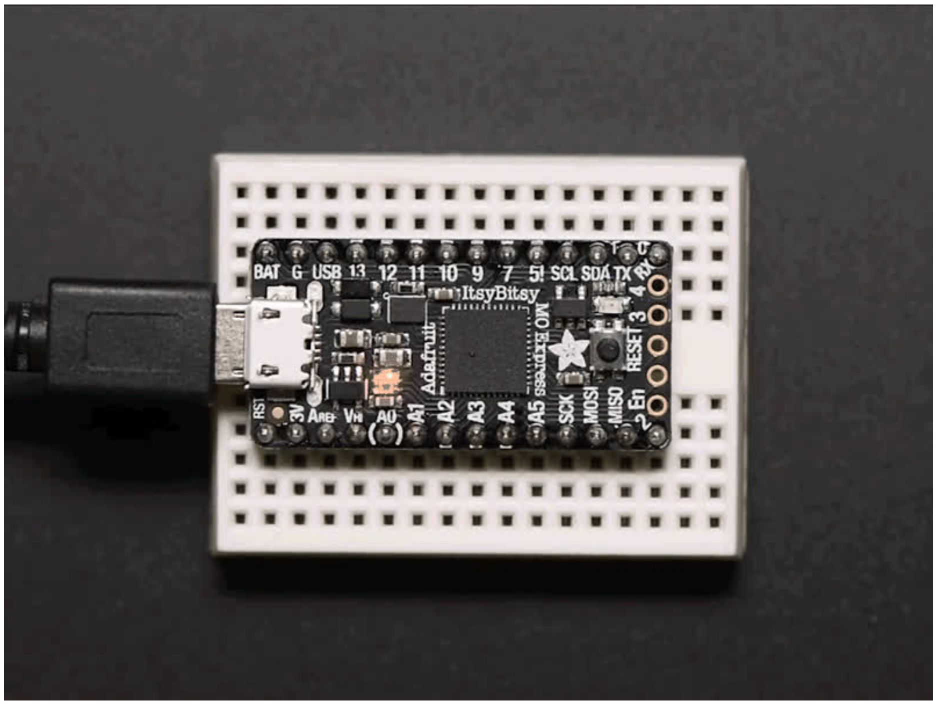
To set the brightness you simply use the `brightness` attribute. Brightness is set with a number between `0` and `1`, representative of a percent from 0% to 100%. So, `led.brightness = (0.3)` sets the LED brightness to 30%. The default brightness is `1` or 100%, and at it's maximum, the LED is blindingly bright! You can set it lower if you choose.

Main Loop

LED colors are set using a combination of **r**ed, **g**reen, and **b**lue, in the form of an (**R**, **G**, **B**) tuple. Each member of the tuple is set to a number between 0 and 255 that determines the amount of each color present. Red, green and blue in different combinations can create all the colors in the rainbow! So, for example, to set the LED to red, the tuple would be (255, 0, 0), which has the maximum level of red, and no green or blue. Green would be (0, 255, 0), etc. For the colors between, you set a combination, such as cyan which is (0, 255, 255), with equal amounts of green and blue.

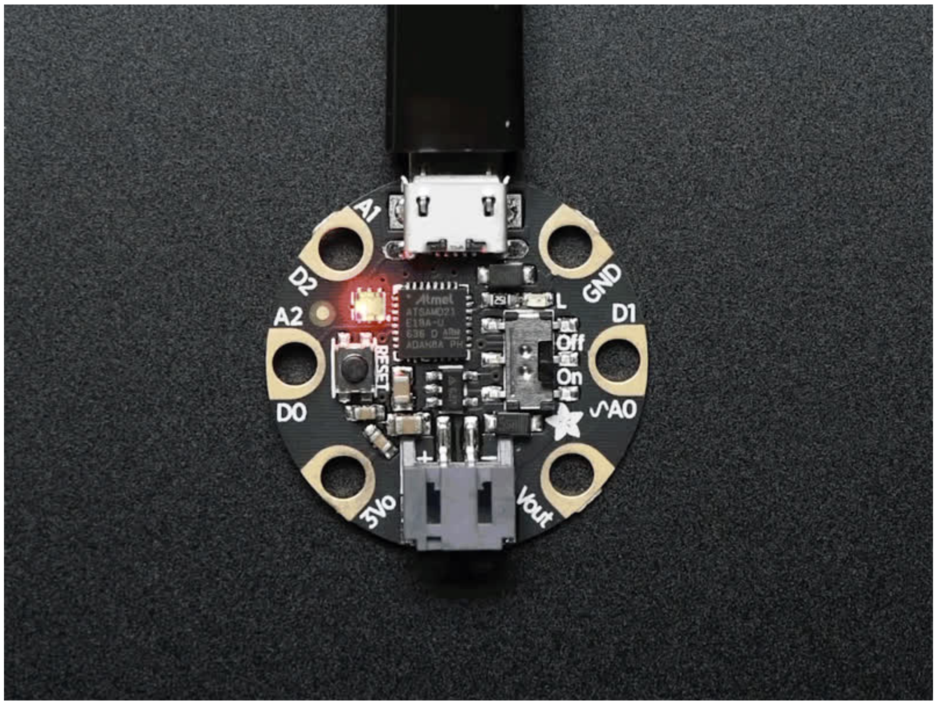
The main loop is quite simple. It sets the first LED to **red** using `(255, 0, 0)`, then **green** using `(0, 255, 0)`, and finally **blue** using `(0, 0, 255)`. Next, we give it a `time.sleep()` so it stays each color for a period of time. We chose `time.sleep(0.5)`, or half a second. Without the `time.sleep()` it'll flash really quickly and the colors will be difficult to see!

Note that we set `led[0]`. This means the first, and in the case of most of the boards, the only LED. In CircuitPython, counting starts at 0. So the first of any object, list, etc will be `0`!



Try changing the numbers in the tuples to change your LED to any color of the rainbow. Or, you can add more lines with different color tuples to add more colors to the sequence. Always add the `time.sleep()`, but try changing the amount of time to create different cycle animations!

Making Rainbows (Because Who Doesn't Love 'Em!)



Coding a rainbow effect involves a little math and a helper function called `wheel`. For details about how wheel works, see this explanation here (https://adafru.it/Bek)!

The last example shows how to do a rainbow animation on the internal RGB LED.

Copy and paste the code into **code.py** using your favorite editor, and save the file. **Remember to comment and uncomment the right lines for the board you're using**, as explained above (https://adafru.it/Bel).

```
import time
import board

# For Trinket M0, Gemma M0, ItsyBitsy M0 Express and ItsyBitsy M4 Express
import adafruit_dotstar
led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
# For Feather M0 Express, Metro M0 Express, Metro M4 Express and Circuit Playground Express
# import neopixel
# led = neopixel.NeoPixel(board.NEOPIXEL, 1)


def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        return 0, 0, 0
    if pos < 85:
        return int(255 - pos * 3), int(pos * 3), 0
    if pos < 170:
        pos -= 85
        return 0, int(255 - pos * 3), int(pos * 3)
    pos -= 170
    return int(pos * 3), 0, int(255 - (pos * 3))


led.brightness = 0.3

i = 0
while True:
    i = (i + 1) % 256  # run from 0 to 255
    led.fill(wheel(i))
    time.sleep(0.1)
```

We add the `wheel` function in after setup but before our main loop.

And right before our main loop, we assign the variable `i = 0`, so it's ready for use inside the loop.

The main loop contains some math that cycles `i` from `0` to `255` and around again repeatedly. We use this value to cycle `wheel()` through the rainbow!

The `time.sleep()` determines the speed at which the rainbow changes. Try a higher number for a slower rainbow or a lower number for a faster one!

Circuit Playground Express Rainbow

Note that here we use `led.fill` instead of `led[0]`. This means it turns on all the LEDs, which in the current code is only one. So why bother with `fill`? Well, you may have a Circuit Playground Express, which as you can see has TEN NeoPixel LEDs built in. The examples so far have only turned on the first one. If you'd like to do a rainbow on all ten LEDs, change the `1` in:

`led = neopixel.NeoPixel(board.NEOPIXEL, 1)`

to `10` so it reads:

`led = neopixel.NeoPixel(board.NEOPIXEL, 10)`.

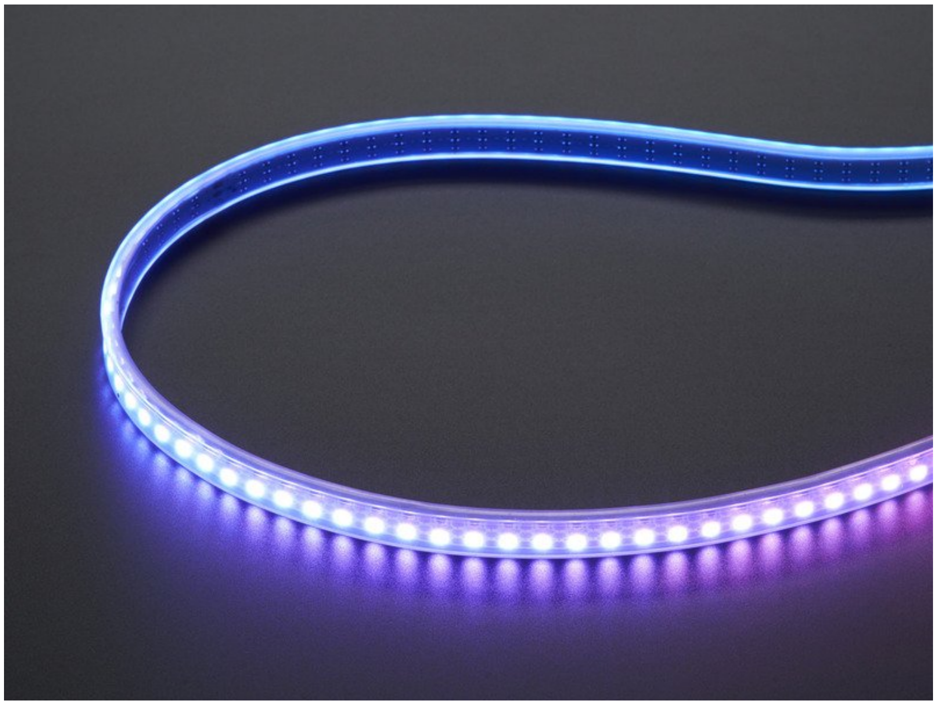
This tells the code to look for 10 LEDs instead of only 1. Now save the code and watch the rainbow go! You can make the same `1` to `10` change to the previous examples as well, and use `led.fill` to light up all the LEDs in the colors you chose! For more details, check out the NeoPixel section of the CPX guide (https://adafru.it/Bem)!

CircuitPython NeoPixel

NeoPixels are a revolutionary and ultra-popular way to add lights and color to your project. These stranded RGB lights have the controller inside the LED, so you just push the RGB data and the LEDs do all the work for you. They're a perfect match for CircuitPython!

You can drive 300 NeoPixel LEDs with brightness control (set `brightness=1.0` in object creation) and 1000 LEDs without. That's because to adjust the brightness we have to dynamically recreate the data-stream each write.

You'll need the **neopixel.mpy** library if you don't already have it in your /lib folder! You can get it from the CircuitPython Library Bundle (https://adafru.it/y8E). If you need help installing the library, check out the CircuitPython Libraries page (https://adafru.it/ABU).



Wiring It Up

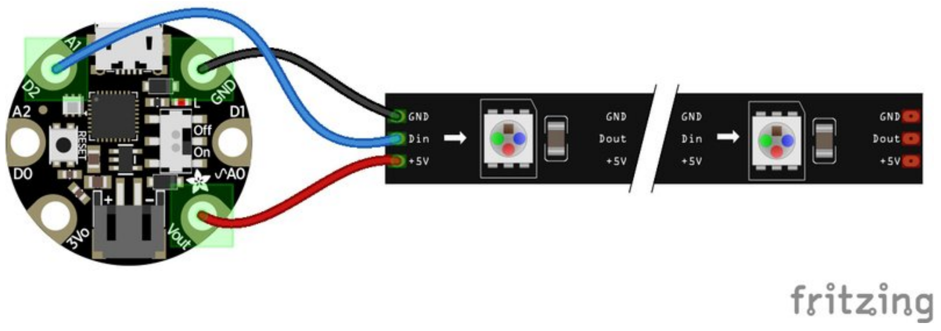
You'll need to solder up your NeoPixels first. Verify your connection is on the **DATA INPUT** or **DIN** side. Plugging into the DATA OUT or DOUT side is a common mistake! The connections are labeled and some formats have arrows to indicate the direction the data must flow.

For powering the pixels from the board, the 3.3V regulator output can handle about 500mA peak which is about 50 pixels with 'average' use. If you want really bright lights and a lot of pixels, we recommend powering direct from an external power source.

- On Gemma M0 and Circuit Playground Express this is the **Vout** pad - that pad has direct power from USB or the battery, depending on which is higher voltage.
- On Trinket M0, Feather M0 Express, Feather M4 Express, ItsyBitsy M0 Express and ItsyBitsy M4 Express the **USB** or **BAT** pins will give you direct power from the USB port or battery.
- On Metro M0 Express and Metro M4 Express, use the **5V** pin regardless of whether it's powered via USB or the DC jack.

If the power to the NeoPixels is greater than 5.5V you may have some difficulty driving some strips, in which case you may need to lower the voltage to 4.5-5V or use a level shifter.

Do not use the VIN pin directly on Metro M0 Express or Metro M4 Express! The voltage can reach 9V and this can destroy your NeoPixels!



Note that the wire ordering on your NeoPixel strip or shape may not exactly match the diagram above. Check the markings to verify which pin is DIN, 5V and GND

The Code

This example includes multiple visual effects. Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython demo - NeoPixel
import time
import board
import neopixel

pixel_pin = board.A1
num_pixels = 8

pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False)


def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        return (0, 0, 0)
    if pos < 85:
        return (255 - pos * 3, pos * 3, 0)
    if pos < 170:
        pos -= 85
        return (0, 255 - pos * 3, pos * 3)
    pos -= 170
    return (pos * 3, 0, 255 - pos * 3)


def color_chase(color, wait):
    for i in range(num_pixels):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
    time.sleep(0.5)


def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = wheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)


RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)

while True:
    pixels.fill(RED)
    pixels.show()
    # Increase or decrease to change the speed of the solid color change.
    time.sleep(1)
    pixels.fill(GREEN)
    pixels.show()
    time.sleep(1)
    pixels.fill(BLUE)
    pixels.show()
    time.sleep(1)

    color_chase(RED, 0.1)  # Increase the number to slow down the color chase
    color_chase(YELLOW, 0.1)
    color_chase(GREEN, 0.1)
    color_chase(CYAN, 0.1)
    color_chase(BLUE, 0.1)
    color_chase(PURPLE, 0.1)

    rainbow_cycle(0)  # Increase the number to slow down the rainbow
```

Create the LED

The first thing we'll do is create the LED object. The NeoPixel object has two required arguments and two optional arguments. You are required to set the pin you're using to drive your NeoPixels and provide the number of pixels you intend to use. You can optionally set `brightness` and `auto_write`.

NeoPixels can be driven by any pin. We've chosen **A1**. To set the pin, assign the variable `pixel_pin` to the pin you'd like to use, in our case `board.A1`.

To provide the number of pixels, assign the variable `num_pixels` to the number of pixels you'd like to use. In this example, we're using a strip of `8`.

We've chosen to set `brightness=0.3`, or 30%.

By default, `auto_write=True`, meaning any changes you make to your pixels will be sent automatically. Since `True` is the default, if you use that setting, you don't need to include it in your LED object at all. We've chosen to set `auto_write=False`. If you set `auto_write=False`, you must include `pixels.show()` each time you'd like to send data to your pixels. This makes your code more complicated, but it can make your LED animations faster!

NeoPixel Helpers

Next we've included a few helper functions to create the super fun visual effects found in this code. First is `wheel()` which we just learned with the Internal RGB LED (https://adafru.it/Bel). Then we have `color_chase()` which requires you to provide a `color` and the amount of time in seconds you'd like between each step of the chase. Next we have `rainbow_cycle()`, which requires you to provide the mount of time in seconds you'd like the animation to take. Last, we've included a list of variables for our colors. This makes it much easier if to reuse the colors anywhere in the code, as well as add more colors for use in multiple places. Assigning and using RGB colors is explained in this section of the CircuitPython Internal RGB LED page (https://adafru.it/Bel).

Main Loop

Thanks to our helpers, our main loop is quite simple. We include the code to set every NeoPixel we're using to red, green and blue for 1 second each. Then we call `color_chase()`, one time for each `color` on our list with `0.1` second delay between setting each subsequent LED the same color during the chase. Last we call `rainbow_cycle(0)`, which means the animation is as fast as it can be. Increase both of those numbers to slow down each animation!

Note that the longer your strip of LEDs, the longer it will take for the animations to complete.

We have a ton more information on general purpose NeoPixel know-how at our NeoPixel UberGuide
https://learn.adafruit.com/adafruit-neopixel-uberguide

NeoPixel RGBW

NeoPixels are available in RGB, meaning there are three LEDs inside, red, green and blue. They're also available in RGBW, which includes four LEDs, red, green, blue and white. The code for RGBW NeoPixels is a little bit different than RGB.

If you run RGB code on RGBW NeoPixels, approximately 3/4 of the LEDs will light up and the LEDs will be the incorrect color even though they may appear to be changing. This is because NeoPixels require a piece of information for each available color (red, green, blue and possibly white).

Therefore, RGB LEDs require three pieces of information and RGBW LEDs require FOUR pieces of information to work. So when you create the LED object for RGBW LEDs, you'll include `bpp=4`, which sets bits-per-pixel to four (the four pieces of information!).

Then, you must include an extra number in every color tuple you create. For example, red will be `(255, 0, 0, 0)`. This is how you send the fourth piece of information. Check out the example below to see how our NeoPixel code looks for using with RGBW LEDs!

```
# CircuitPython demo - NeoPixel RGBW

import time
import board
import neopixel
```

```
pixel_pin = board.A1
num_pixels = 8

pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False,
                           pixel_order=(1, 0, 2, 3))


def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        return (0, 0, 0, 0)
    if pos < 85:
        return (255 - pos * 3, pos * 3, 0, 0)
    if pos < 170:
        pos -= 85
        return (0, 255 - pos * 3, pos * 3, 0)
    pos -= 170
    return (pos * 3, 0, 255 - pos * 3, 0)


def color_chase(color, wait):
    for i in range(num_pixels):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
    time.sleep(0.5)


def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = wheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)


RED = (255, 0, 0, 0)
YELLOW = (255, 150, 0, 0)
GREEN = (0, 255, 0, 0)
CYAN = (0, 255, 255, 0)
BLUE = (0, 0, 255, 0)
PURPLE = (180, 0, 255, 0)

while True:
    pixels.fill(RED)
    pixels.show()
    # Increase or decrease to change the speed of the solid color change.
    time.sleep(1)
    pixels.fill(GREEN)
    pixels.show()
    time.sleep(1)
    pixels.fill(BLUE)
    pixels.show()
    time.sleep(1)

    color_chase(RED, 0.1)  # Increase the number to slow down the color chase
    color_chase(YELLOW, 0.1)
    color_chase(GREEN, 0.1)
    color_chase(CYAN, 0.1)
    color_chase(BLUE, 0.1)
    color_chase(PURPLE, 0.1)

    rainbow_cycle(0)  # Increase the number to slow down the rainbow
```

Read the Docs

For a more in depth look at what `neopixel` can do, check out NeoPixel on Read the Docs (https://adafru.it/C5m).

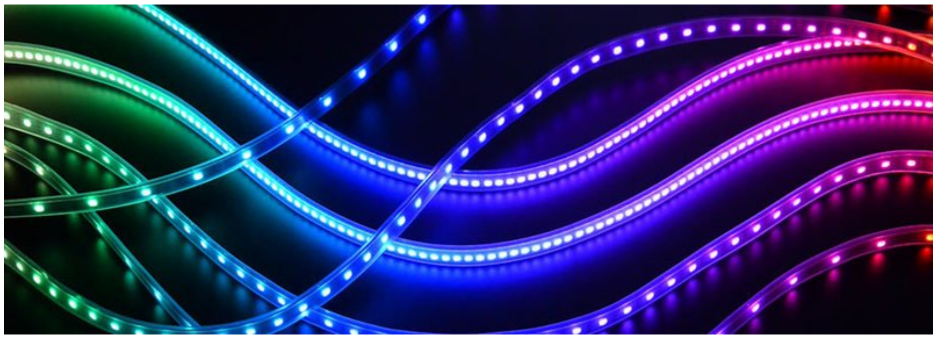
CircuitPython DotStar

DotStars use two wires, unlike NeoPixel's one wire. They're very similar but you can write to DotStars much faster with hardware SPI *and* they have a faster PWM cycle so they are better for light painting.

Any pins can be used **but** if the two pins can form a hardware SPI port, the library will automatically switch over to hardware SPI. If you use hardware SPI then you'll get 4 MHz clock rate (that would mean updating a 64 pixel strand in about 500uS - that's 0.0005 seconds). If you use non-hardware SPI pins you'll drop down to about 3KHz, 1000 times as slow!

You can drive 300 DotStar LEDs with brightness control (set `brightness=1.0` in object creation) and 1000 LEDs without. That's because to adjust the brightness we have to dynamically recreate the data-stream each write.

You'll need the **adafruit_dotstar.mpy** library if you don't already have it in your /lib folder! You can get it from the CircuitPython Library Bundle (https://adafru.it/y8E). If you need help installing the library, check out the CircuitPython Libraries page (https://adafru.it/ABU).



Wire It Up

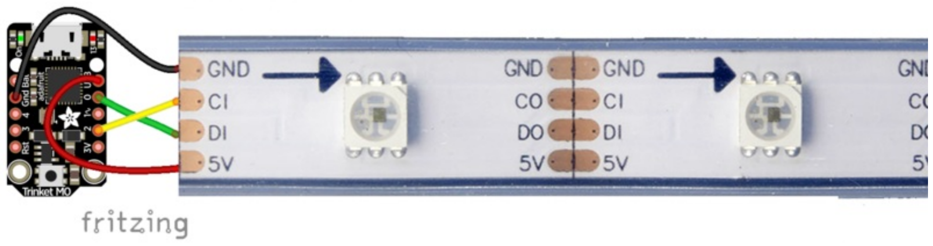
You'll need to solder up your DotStars first. Verify your connection is on the **DATA INPUT** or **DI** and **CLOCK INPUT** or **CI** side. Plugging into the DATA OUT/DO or CLOCK OUT/CO side is a common mistake! The connections are labeled and some formats have arrows to indicate the direction the data must flow. Always verify your wiring with a visual inspection, as the order of the connections can differ from strip to strip!

For powering the pixels from the board, the 3.3V regulator output can handle about 500mA peak which is about 50 pixels with 'average' use. If you want really bright lights and a lot of pixels, we recommend powering direct from an external power source.

- On Gemma M0 and Circuit Playground Express this is the **Vout** pad - that pad has direct power from USB or the battery, depending on which is higher voltage.
- On Trinket M0, Feather M0 Express, Feather M4 Express, ItsyBitsy M0 Express and ItsyBitsy M4 Express the **USB** or **BAT** pins will give you direct power from the USB port or battery.
- On Metro M0 Express and Metro M4 Express, use the **5V** pin regardless of whether it's powered via USB or the DC jack.

If the power to the DotStars is greater than 5.5V you may have some difficulty driving some strips, in which case you may need to lower the voltage to 4.5-5V or use a level shifter.

Do not use the VIN pin directly on Metro M0 Express or Metro M4 Express! The voltage can reach 9V and this can destroy your DotStars!



Note that the wire ordering on your DotStar strip or shape may not exactly match the diagram above. Check the markings to verify which pin is DIN, CIN, 5V and GND

The Code

This example includes multiple visual effects. Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython demo - Dotstar
import time
import adafruit_dotstar
import board

num_pixels = 30
pixels = adafruit_dotstar.DotStar(board.A1, board.A2, num_pixels, brightness=0.1, auto_write=False)


def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        return (0, 0, 0)
    if pos < 85:
        return (255 - pos * 3, pos * 3, 0)
    if pos < 170:
        pos -= 85
        return (0, 255 - pos * 3, pos * 3)
    pos -= 170
    return (pos * 3, 0, 255 - pos * 3)


def color_fill(color, wait):
    pixels.fill(color)
    pixels.show()
    time.sleep(wait)


def slice_alternating(wait):
    pixels[::2] = [RED] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [ORANGE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [YELLOW] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [GREEN] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [TEAL] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [CYAN] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [BLUE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [PURPLE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [MAGENTA] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [WHITE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)


def slice_rainbow(wait):
    pixels[::6] = [RED] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[1::6] = [ORANGE] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[2::6] = [YELLOW] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[3::6] = [GREEN] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[4::6] = [BLUE] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[5::6] = [PURPLE] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)


def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = wheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)


RED = (255, 0, 0)
YELLOW = (255, 150, 0)
```

```
ORANGE = (255, 40, 0)
GREEN = (0, 255, 0)
TEAL = (0, 255, 120)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
MAGENTA = (255, 0, 20)
WHITE = (255, 255, 255)

while True:
    # Change this number to change how long it stays on each solid color.
    color_fill(RED, 0.5)
    color_fill(YELLOW, 0.5)
    color_fill(ORANGE, 0.5)
    color_fill(GREEN, 0.5)
    color_fill(TEAL, 0.5)
    color_fill(CYAN, 0.5)
    color_fill(BLUE, 0.5)
    color_fill(PURPLE, 0.5)
    color_fill(MAGENTA, 0.5)
    color_fill(WHITE, 0.5)

    # Increase or decrease this to speed up or slow down the animation.
    slice_alternating(0.1)

    color_fill(WHITE, 0.5)

    # Increase or decrease this to speed up or slow down the animation.
    slice_rainbow(0.1)

    time.sleep(0.5)

    # Increase this number to slow down the rainbow animation.
    rainbow_cycle(0)
```

We've chosen pins A1 and A2, but these are not SPI pins on all boards. DotStars respond faster when using hardware SPI!

Create the LED

The first thing we'll do is create the LED object. The DotStar object has three required arguments and two optional arguments. You are required to set the pin you're using for data, set the pin you'll be using for clock, and provide the number of pixels you intend to use. You can optionally set `brightness` and `auto_write`.

DotStars can be driven by any two pins. We've chosen **A1** for clock and **A2** for data. To set the pins, include the pin names at the beginning of the object creation, in this case `board.A1` and `board.A2`.

To provide the number of pixels, assign the variable `num_pixels` to the number of pixels you'd like to use. In this example, we're using a strip of `72`.

We've chosen to set `brightness=0.1`, or 10%.

By default, `auto_write=True`, meaning any changes you make to your pixels will be sent automatically. Since `True` is the default, if you use that setting, you don't need to include it in your LED object at all. We've chosen to set `auto_write=False`. If you set `auto_write=False`, you must include `pixels.show()` each time you'd like to send data to your pixels. This makes your code more complicated, but it can make your LED animations faster!

DotStar Helpers

We've included a few helper functions to create the super fun visual effects found in this code.

First is `wheel()` which we just learned with the Internal RGB LED (https://adafru.it/Bel). Then we have `color_fill()` which requires you to provide a `color` and the length of time you'd like it to be displayed. Next, are `slice_alternating()`, `slice_rainbow()`, and `rainbow_cycle()` which require you to provide the amount of time in seconds you'd between each step of the animation.

Last, we've included a list of variables for our colors. This makes it much easier if to reuse the colors anywhere in the code, as well as add more colors for use in multiple places. Assigning and using RGB colors is explained in this section of the CircuitPython Internal RGB LED page (https://adafru.it/Bel).

The two slice helpers utilise a nifty feature of the DotStar library that allows us to use math to light up LEDs in repeating patterns. `slice_alternating()` first lights up the even number LEDs and then the odd number LEDs and repeats this back and forth. `slice_rainbow()` lights up every sixth LED with one of the six rainbow colors until the strip is filled. Both use our handy color variables. This slice code only works when the total number of LEDs is divisible by the slice size, in our case 2 and 6. DotStars come in strips of 30, 60, 72, and 144, all of which are divisible by 2 and 6. In the event that you cut them into different sized strips, the code in this example may not work without modification. However, as long as you provide a total number of LEDs that is divisible by the slices, the code will work.

Main Loop

Our main loop begins by calling `color_fill()` once for each `color` on our list and sets each to hold for `0.5` seconds. You can change this number to change how fast each color is displayed. Next, we call `slice_alternating(0.1)`, which means there's a 0.1 second delay between each change in the animation. Then, we fill the strip white to create a clean backdrop for the rainbow to display. Then, we call `slice_rainbow(0.1)`, for a 0.1 second delay in the animation. Last we call `rainbow_cycle(0)`, which means it's as fast as it can possibly be. Increase or decrease either of these numbers to speed up or slow down the animations!

Note that the longer your strip of LEDs is, the longer it will take for the animations to complete.

We have a ton more information on general purpose DotStar know-how at our DotStar UberGuide
https://learn.adafruit.com/adafruit-dotstar-leds

Is it SPI?

We explained at the beginning of this section that the LEDs respond faster if you're using hardware SPI. On some of the boards, there are HW SPI pins directly available in the form of MOSI and SCK. However, hardware SPI is available on more than just those pins. But, how can you figure out which? Easy! We wrote a handy script.

We chose pins **A1** and **A2** for our example code. To see if these are hardware SPI on the board you're using, copy and paste the code into **code.py** using your favorite editor, and save the file. Then connect to the serial console to see the results.

To check if other pin combinations have hardware SPI, change the pin names on the line reading: `if is_hardware_SPI(board.A1, board.A2):` to the pins you want to use. Then, check the results in the serial console. Super simple!

```
import board
import busio


def is_hardware_spi(clock_pin, data_pin):
    try:
        p = busio.SPI(clock_pin, data_pin)
        p.deinit()
        return True
    except ValueError:
        return False


# Provide the two pins you intend to use.
if is_hardware_spi(board.A1, board.A2):
    print("This pin combination is hardware SPI!")
else:
    print("This pin combination isn't hardware SPI.")
```

Read the Docs

For a more in depth look at what `dotstar` can do, check out DotStar on Read the Docs (https://adafru.it/C4d).

CircuitPython UART Serial

In addition to the USB-serial connection you use for the REPL, there is also a *hardware* UART you can use. This is handy to talk to UART devices like GPSs, some sensors, or other microcontrollers!

This quick-start example shows how you can create a UART device for communicating with hardware serial devices.

To use this example, you'll need something to generate the UART data. We've used a GPS! Note that the GPS will give you UART data without getting a fix on your location. You can use this example right from your desk! You'll have data to read, it simply won't include your actual location.

You'll need the **adafruit_bus_device** library folder if you don't already have it in your /lib folder! You can get it from the CircuitPython Library Bundle (https://adafru.it/y8E). If you need help installing the library, check out the CircuitPython Libraries page (https://adafru.it/ABU).

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython Demo - USB/Serial echo

import board
import busio
import digitalio

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

uart = busio.UART(board.TX, board.RX, baudrate=9600)

while True:
    data = uart.read(32)  # read up to 32 bytes
    # print(data)  # this is a bytearray type

    if data is not None:
        led.value = True

        # convert bytearray to string
        data_string = ''.join([chr(b) for b in data])
        print(data_string, end="")

        led.value = False
```

The Code

First we create the UART object. We provide the pins we'd like to use, `board.TX` and `board.RX`, and we set the `baudrate=9600`. While these pins are labeled on most of the boards, be aware that RX and TX are not labeled on Gemma, and are labeled on the bottom of Trinket. See the diagrams below for help with finding the correct pins on your board.

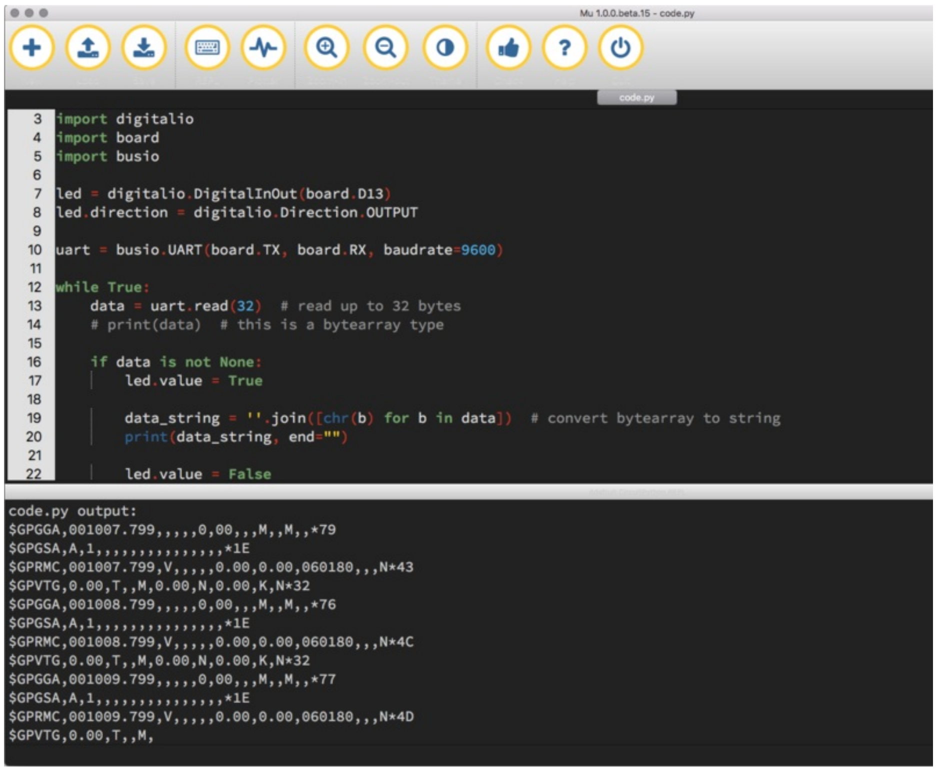
Once the object is created you read data in with `read(`*numbytes*`)` where you can specify the max number of bytes. It will return a byte array type object if anything was received already. Note it will always return immediately because there is an internal buffer! So read as much data as you can 'digest'.

If there is no data available, `read()` will return `None`, so check for that before continuing.

The data that is returned is in a byte array, if you want to convert it to a string, you can use this handy line of code which will run `chr()` on each byte:

`datastr = ''.join([chr(b) for b in data]) # convert bytearray to string`

Your results will look something like this:



```
import digitalio
import board
import busio

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

uart = busio.UART(board.TX, board.RX, baudrate=9600)

while True:
    data = uart.read(32)  # read up to 32 bytes
    # print(data)  # this is a bytearray type

    if data is not None:
        led.value = True

        data_string = ''.join([chr(b) for b in data])  # convert bytearray to string
        print(data_string, end="")

        led.value = False
```

```
code.py output:
$GPGGA,001007.799,,,,,0,00,,,M,,M,,*79
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,001007.799,V,,,,,0.00,0.00,060180,,,N*43
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,001008.799,,,,,0,00,,,M,,M,,*76
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,001008.799,V,,,,,0.00,0.00,060180,,,N*4C
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,001009.799,,,,,0,00,,,M,,M,,*77
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,001009.799,V,,,,,0.00,0.00,060180,,,N*4D
$GPVTG,0.00,T,,M,
```

For more information about the data you're reading and the Ultimate GPS, check out the Ultimate GPS guide: https://learn.adafruit.com/adafruit-ultimate-gps

Wire It Up

You'll need a couple of things to connect the GPS to your board.

For Gemma M0 and Circuit Playground Express, you can use use alligator clips to connect to the Flora Ultimate GPS Module.

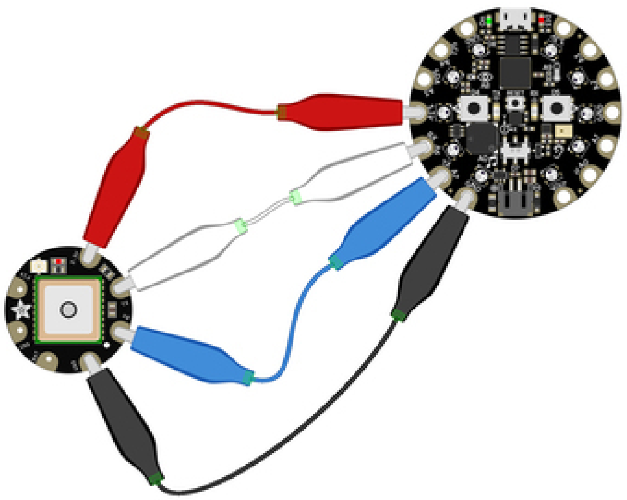
For Trinket M0, Feather M0 Express, Metro M0 Express and ItsyBitsy M0 Express, you'll need a breadboard and jumper wires to connect to the Ultimate GPS Breakout.

We've included diagrams show you how to connect the GPS to your board. In these diagrams, the wire colors match the same pins on each board.

- The **black** wire connects between the **ground** pins.
- The **red** wire connects between the **power** pins on the GPS and your board.
- The **blue** wire connects from **TX** on the GPS to **RX** on your board.
- The **white** wire connects from **RX** on the GPS to **TX** on your board.

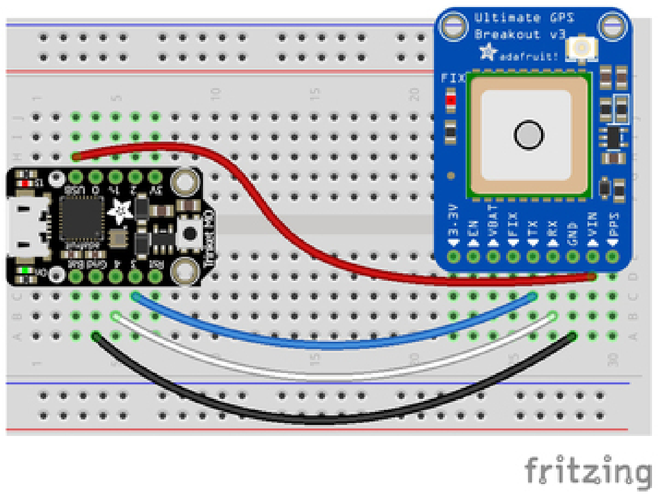
Check out the list below for a diagram of your specific board!

Watch out! A common mixup with UART serial is that RX on one board connects to TX on the other! However, sometimes boards have RX labeled TX and vice versa. So, you'll want to start with RX connected to TX, but if that doesn't work, try the other way around!



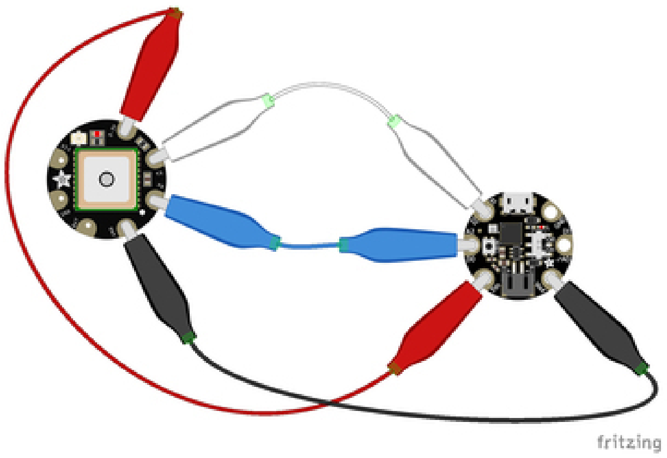
Circuit Playground Express and Circuit Playground Bluefruit

- Connect **3.3v** on your CPX to **3.3v** on your GPS.
- Connect **GND** on your CPX to **GND** on your GPS.
- Connect **RX/A6** on your CPX to **TX** on your GPS.
- Connect **TX/A7** on your CPX to **RX** on your GPS.



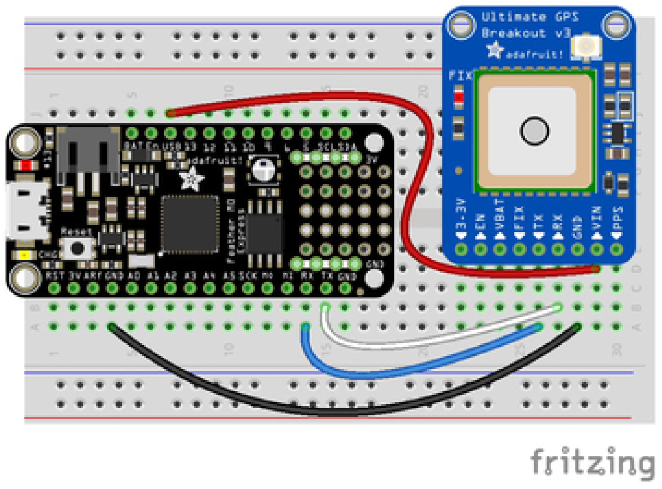
Trinket M0

- Connect **USB** on the Trinket to **VIN** on the GPS.
- Connect **Gnd** on the Trinket to **GND** on the GPS.
- Connect **D3** on the Trinket to **TX** on the GPS.
- Connect **D4** on the Trinket to **RX** on the GPS.



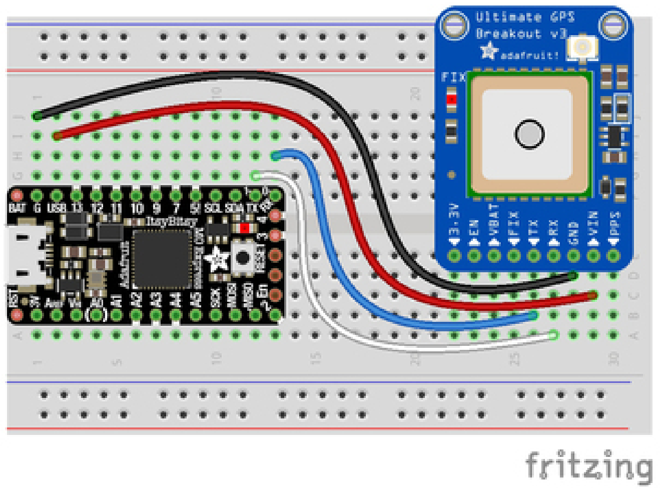
Gemma M0

- Connect **3vo** on the Gemma to **3.3v** on the GPS.
- Connect **GND** on the Gemma to **GND** on the GPS.
- Connect **A1/D2** on the Gemma to **TX** on the GPS.
- Connect **A2/D0** on the Gemma to **RX** on the GPS.



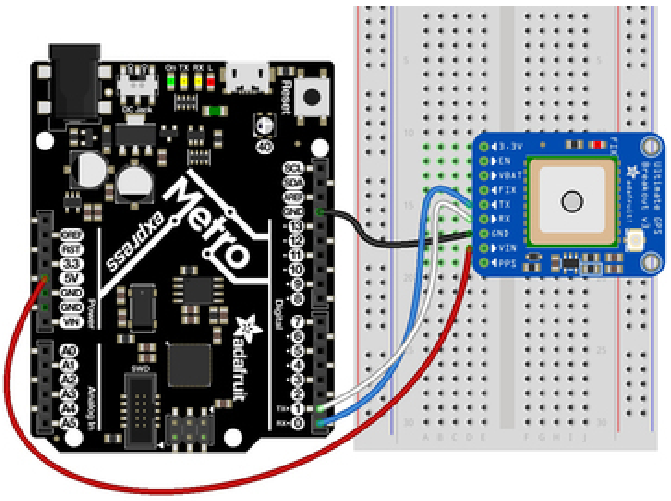
Feather M0 Express and Feather M4 Express

- Connect **USB** on the Feather to **VIN** on the GPS.
- Connect **GND** on the Feather to **GND** on the GPS.
- Connect **RX** on the Feather to **TX** on the GPS.
- Connect **TX** on the Feather to **RX** on the GPS.



ItsyBitsy M0 Express and ItsyBitsy M4 Express

- Connect **USB** on the ItsyBitsy to **VIN** on the GPS
- Connect **G** on the ItsyBitsy to **GND** on the GPS.
- Connect **RX/0** on the ItsyBitsy to **TX** on the GPS.
- Connect **TX/1** on the ItsyBitsy to **RX** on the GPS.



Metro M0 Express and Metro M4 Express

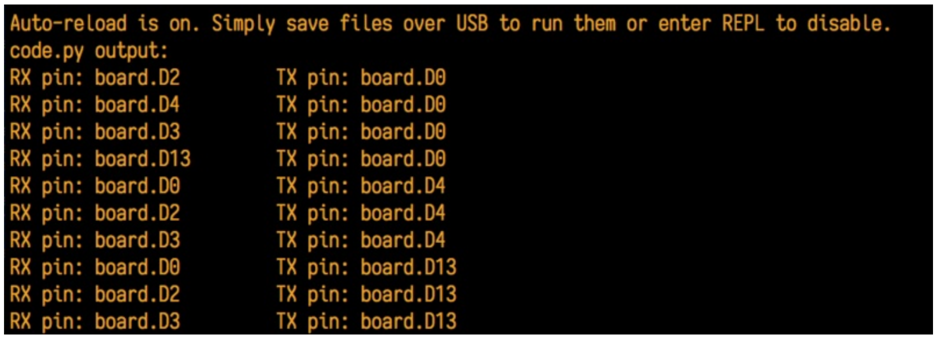
- Connect **5V** on the Metro to **VIN** on the GPS.
- Connect **GND** on the Metro to **GND** on the GPS.
- Connect **RX/D0** on the Metro to **TX** on the GPS.
- Connect **TX/D1** on the Metro to **RX** on the GPS.

Where's my UART?

On the SAMD21, we have the flexibility of using a wide range of pins for UART. Compare this to some chips like the ESP8266 with *fixed* UART pins. The good news is you can use many but not *all* pins. Given the large number of SAMD boards we have, its impossible to guarantee anything other than the labeled 'TX' and 'RX'. So, if you want some other setup, or multiple UARTs, how will you find those pins? Easy! We've written a handy script.

All you need to do is copy this file to your board, rename it **code.py**, connect to the serial console and check out the output! The results print out a nice handy list of RX and TX pin pairs that you can use.

These are the results from a Trinket M0, your output may vary and it might be *very* long. For more details about UARTs and SERCOMs check out our detailed guide here (https://adafru.it/Ben)



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
RX pin: board.D2        TX pin: board.D0
RX pin: board.D4        TX pin: board.D0
RX pin: board.D3        TX pin: board.D0
RX pin: board.D13       TX pin: board.D0
RX pin: board.D0        TX pin: board.D4
RX pin: board.D2        TX pin: board.D4
RX pin: board.D3        TX pin: board.D4
RX pin: board.D0        TX pin: board.D13
RX pin: board.D2        TX pin: board.D13
RX pin: board.D3        TX pin: board.D13
```

```
import board
import busio
from microcontroller import Pin


def is_hardware_uart(tx, rx):
    try:
        p = busio.UART(tx, rx)
        p.deinit()
        return True
    except ValueError:
        return False


def get_unique_pins():
    exclude = ['NEOPIXEL', 'APA102_MOSI', 'APA102_SCK']
    pins = [pin for pin in [
        getattr(board, p) for p in dir(board) if p not in exclude]
            if isinstance(pin, Pin)]
    unique = []
    for p in pins:
        if p not in unique:
            unique.append(p)
    return unique


for tx_pin in get_unique_pins():
    for rx_pin in get_unique_pins():
        if rx_pin is tx_pin:
            continue
        else:
            if is_hardware_uart(tx_pin, rx_pin):
                print("RX pin:", rx_pin, "\t TX pin:", tx_pin)
            else:
                pass
```

Trinket M0: Create UART before I2C

On the Trinket M0 (only), if you are using both `busio.UART` and `busio.I2C`, you must create the UART object first, e.g.:

```
>>> import board,busio
>>> uart = busio.UART(board.TX, board.RX)
>>> i2c = busio.I2C(board.SCL, board.SDA)
```

Creating `busio.I2C` first does not work:

```
>>> import board,busio
>>> i2c = busio.I2C(board.SCL, board.SDA)
>>> uart = busio.UART(board.TX, board.RX)
Traceback (most recent call last):
File "", line 1, in
ValueError: Invalid pins
```

CircuitPython I2C

I2C is a 2-wire protocol for communicating with simple sensors and devices, meaning it uses two connections for transmitting and receiving data. There are many I2C devices available and they're really easy to use with CircuitPython. We have libraries available for many I2C devices in the library bundle (https://adafru.it/uap). (If you don't see the sensor you're looking for, keep checking back, more are being written all the time!)

In this section, we're going to do is learn how to scan the I2C bus for all connected devices. Then we're going to learn how to interact with an I2C device.

We'll be using the TSL2561, a common, low-cost light sensor. While the exact code we're running is specific to the TSL2561 the overall process is the same for just about any I2C sensor or device.

You'll need the **adafruit_tsl2561.mpy** library and **adafruit_bus_device** library folder if you don't already have it in your /lib folder! You can get it from the CircuitPython Library Bundle (https://adafru.it/y8E). If you need help installing the library, check out the CircuitPython Libraries page (https://adafru.it/ABU).

These examples will use the TSL2561 lux sensor Flora and breakout. The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.

Wire It Up

You'll need a couple of things to connect the TSL2561 to your board.

For Gemma M0 and Circuit Playground Express, you can use use alligator clips to connect to the Flora TSL2561 Lux Sensor.

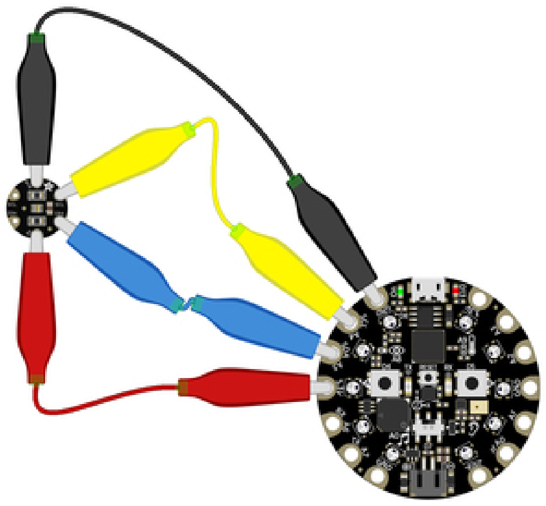
For Trinket M0, Feather M0 Express, Metro M0 Express and ItsyBitsy M0 Express, you'll need a breadboard and jumper wires to connect to the TSL2561 Lux Sensor breakout board.

We've included diagrams show you how to connect the TSL2561 to your board. In these diagrams, the wire colors match the same pins on each board.

- The **black** wire connects between the **ground** pins.
- The **red** wire connects between the **power** pins on the TSL2561 and your board.
- The **yellow** wire connects from **SCL** on the TSL2561 to **SCL** on your board.
- The **blue** wire connects from **SDA** on the TSL2561 to **SDA** on your board.

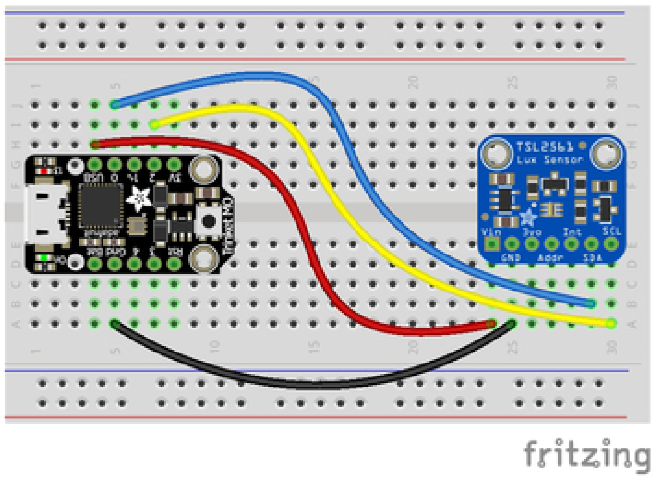
Check out the list below for a diagram of your specific board!

Be aware that the Adafruit microcontroller boards do not have I2C pullup resistors built in! All of the Adafruit breakouts do, but if you're building your own board or using a non-Adafruit breakout, you must add 2.2K-10K ohm pullups on both SDA and SCL to the 3.3V.



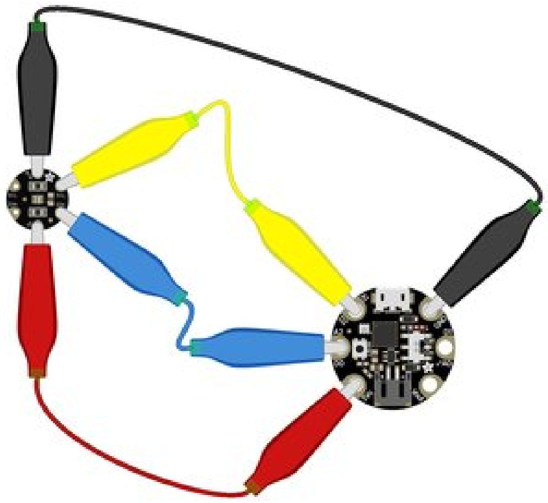
Circuit Playground Express and Circuit Playground Bluefruit

- Connect **3.3v** on your CPX to **3.3v** on your TSL2561.
- Connect **GND** on your CPX to **GND** on your TSL2561.
- Connect **SCL/A4** on your CPX to **SCL** on your TSL2561.
- Connect **SDL/A5** on your CPX to **SDA** on your TSL2561.



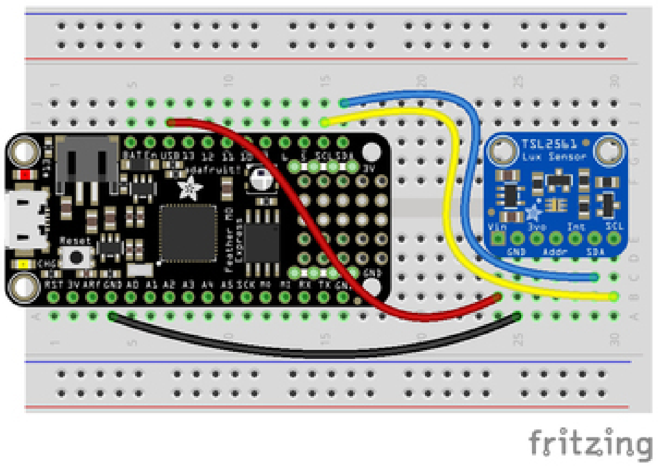
Trinket M0

- Connect **USB** on the Trinket to **VIN** on the TSL2561.
- Connect **Gnd** on the Trinket to **GND** on the TSL2561.
- Connect **D2** on the Trinket to **SCL** on the TSL2561.
- Connect **D0** on the Trinket to **SDA** on the TSL2561.



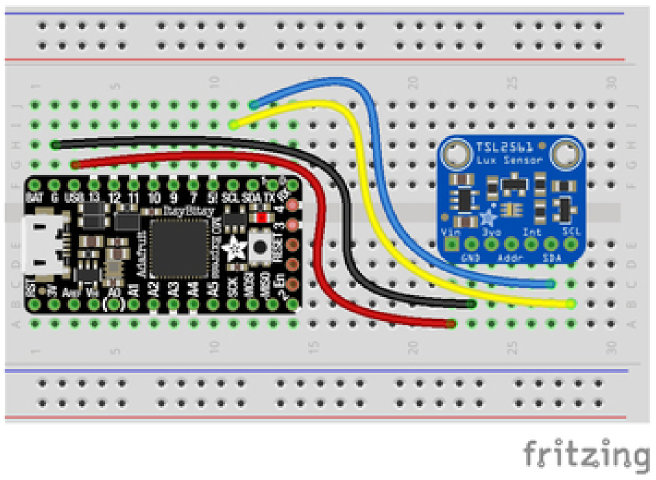
Gemma M0

- Connect **3vo** on the Gemma to **3V** on the TSL2561.
- Connect **GND** on the Gemma to **GND** on the TSL2561.
- Connect **A1/D2** on the Gemma to **SCL** on the TSL2561.
- Connect **A2/D0** on the Gemma to **SDA** on the TSL2561.



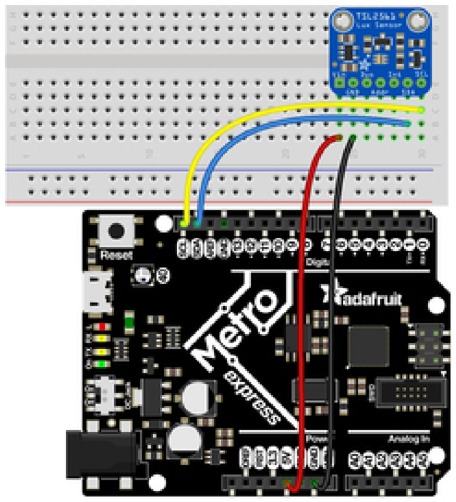
Feather M0 Express and Feather M4 Express

- Connect **USB** on the Feather to **VIN** on the TSL2561.
- Connect **GND** on the Feather to **GND** on the TSL2561.
- Connect **SCL** on the Feather to **SCL** on the TSL2561.
- Connect **SDA** on the Feather to **SDA** on the TSL2561.



ItsyBitsy M0 Express and ItsyBitsy M4 Express

- Connect **USB** on the ItsyBitsy to **VIN** on the TSL2561
- Connect **G** on the ItsyBitsy to **GND** on the TSL2561.
- Connect **SCL** on the ItsyBitsy to **SCL** on the TSL2561.
- Connect **SDA** on the ItsyBitsy to **SDA** on the TSL2561.



Metro M0 Express and Metro M4 Express

- Connect **5V** on the Metro to **VIN** on the TSL2561.
- Connect **GND** on the Metro to **GND** on the TSL2561.
- Connect **SCL** on the Metro to **SCL** on the TSL2561.
- Connect **SDA** on the Metro to **SDA** on the TSL2561.

Find Your Sensor

The first thing you'll want to do after getting the sensor wired up, is make sure it's wired correctly. We're going to do an I2C scan to see if the board is detected, and if it is, print out its I2C address.

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython demo - I2C scan

import time

import board
import busio

i2c = busio.I2C(board.SCL, board.SDA)

while not i2c.try_lock():
    pass

while True:
    print("I2C addresses found:", [hex(device_address)
                                   for device_address in i2c.scan()])
    time.sleep(2)
```

First we create the `i2c` object and provide the I2C pins, `board.SCL` and `board.SDA`.

To be able to scan it, we need to lock the I2C down so the only thing accessing it is the code. So next we include a loop that waits until I2C is locked and then continues on to the scan function.

Last, we have the loop that runs the actual scan, `i2c_scan()`. Because I2C typically refers to addresses in hex form, we've included this bit of code that formats the results into hex format: `[hex(device_address) for device_address in` i2c.scan()] .

Open the serial console to see the results! The code prints out an array of addresses. We've connected the TSL2561 which has a 7-bit I2C address of 0x39. The result for this sensor is I2C addresses found: ['0x39'] . If no addresses are returned, refer back to the wiring diagrams to make sure you've wired up your sensor correctly.

I2C Sensor Data

Now we know for certain that our sensor is connected and ready to go. Let's find out how to get the data from our sensor!

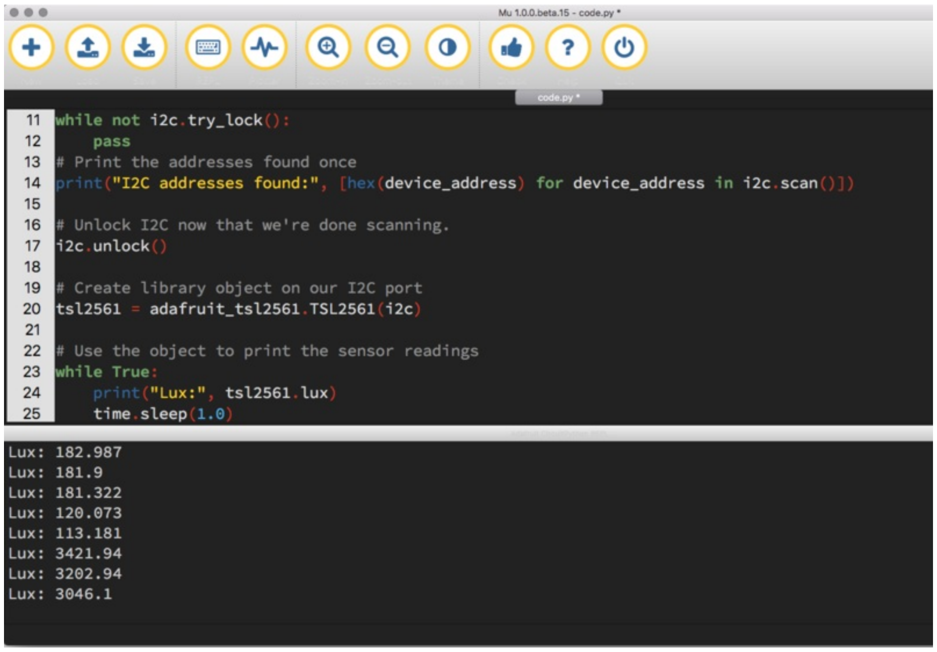
Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython Demo - I2C sensor

import time

import adafruit_tsl2561
import board
import busio

i2c = busio.I2C(board.SCL, board.SDA)

# Lock the I2C device before we try to scan
while not i2c.try_lock():
    pass
# Print the addresses found once
print("I2C addresses found:", [hex(device_address)
                               for device_address in i2c.scan()])

# Unlock I2C now that we're done scanning.
i2c.unlock()

# Create library object on our I2C port
tsl2561 = adafruit_tsl2561.TSL2561(i2c)

# Use the object to print the sensor readings
while True:
    print("Lux:", tsl2561.lux)
    time.sleep(1.0)
```

This code begins the same way as the scan code. We've included the scan code so you have verification that your sensor is wired up correctly and is detected. It prints the address once. After the scan, we unlock I2C with i2c_unlock() so we can use the sensor for data.

We create our sensor object using the sensor library. We call it tsl2561 and provide it the i2c object.

Then we have a simple loop that prints out the lux reading using the sensor object we created. We add a time.sleep(1.0) , so it only prints once per second. Connect to the serial console to see the results. Try shining a light on it to see the results change!



```
while not i2c.try_lock():
    pass
# Print the addresses found once
print("I2C addresses found:", [hex(device_address) for device_address in i2c.scan()])

# Unlock I2C now that we're done scanning.
i2c.unlock()

# Create library object on our I2C port
tsl2561 = adafruit_tsl2561.TSL2561(i2c)

# Use the object to print the sensor readings
while True:
    print("Lux:", tsl2561.lux)
    time.sleep(1.0)
```

```
Lux: 182.987
Lux: 181.9
Lux: 181.322
Lux: 120.073
Lux: 113.181
Lux: 3421.94
Lux: 3202.94
Lux: 3046.1
```

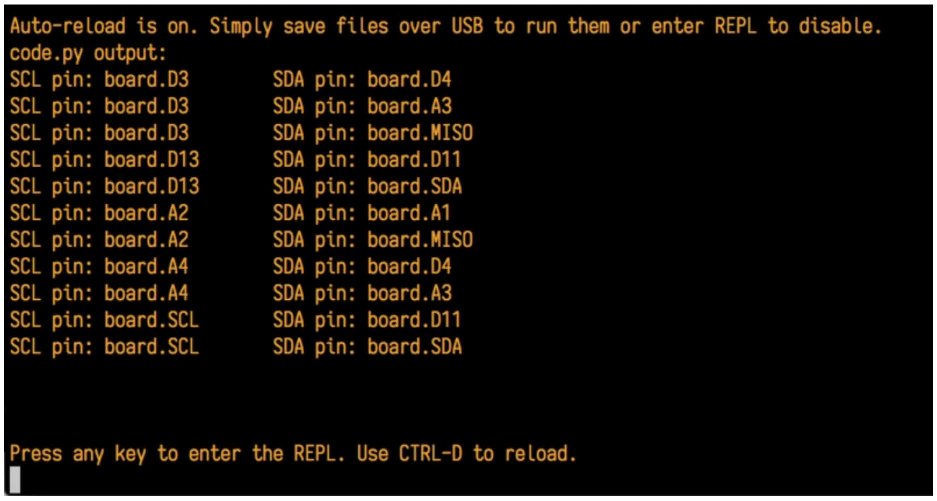
Where's my I2C?

On the SAMD21, SAMD51 and nRF52840, we have the flexibility of using a wide range of pins for I2C. On the nRF52840, any pin can be used for I2C! Some chips, like the ESP8266, require using bitbangio, but can also use any pins for I2C. There's some other chips that may have fixed I2C pin.

The good news is you can use many but not *all* pins. Given the large number of SAMD boards we have, its impossible to guarantee anything other than the labeled 'SDA' and 'SCL'. So, if you want some other setup, or multiple I2C interfaces, how will you find those pins? Easy! We've written a handy script.

All you need to do is copy this file to your board, rename it **code.py**, connect to the serial console and check out the output! The results print out a nice handy list of SCL and SDA pin pairs that you can use.

These are the results from an ItsyBitsy M0 Express. Your output may vary and it might be *very* long. For more details about I2C and SERCOMs, check out our detailed guide here (https://adafru.it/Ben).



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
SCL pin: board.D3       SDA pin: board.D4
SCL pin: board.D3       SDA pin: board.A3
SCL pin: board.D3       SDA pin: board.MISO
SCL pin: board.D13      SDA pin: board.D11
SCL pin: board.D13      SDA pin: board.SDA
SCL pin: board.A2       SDA pin: board.A1
SCL pin: board.A2       SDA pin: board.MISO
SCL pin: board.A4       SDA pin: board.D4
SCL pin: board.A4       SDA pin: board.A3
SCL pin: board.SCL      SDA pin: board.D11
SCL pin: board.SCL      SDA pin: board.SDA



Press any key to enter the REPL. Use CTRL-D to reload.
```

```
import board
import busio
from microcontroller import Pin

def is_hardware_I2C(scl, sda):
    try:
        p = busio.I2C(scl, sda)
        p.deinit()
        return True
    except ValueError:
        return False
    except RuntimeError:
        return True


def get_unique_pins():
    exclude = ['NEOPIXEL', 'APA102_MOSI', 'APA102_SCK']
    pins = [pin for pin in [
        getattr(board, p) for p in dir(board) if p not in exclude]
            if isinstance(pin, Pin)]
    unique = []
    for p in pins:
        if p not in unique:
            unique.append(p)
    return unique


for scl_pin in get_unique_pins():
    for sda_pin in get_unique_pins():
        if scl_pin is sda_pin:
            continue
        else:
            if is_hardware_I2C(scl_pin, sda_pin):
                print("SCL pin:", scl_pin, "\t SDA pin:", sda_pin)
            else:
                pass
```

CircuitPython HID Keyboard and Mouse

These examples have been updated for version 4+ of the CircuitPython HID library. On some boards, such as the CircuitPlayground Express, this library is built into CircuitPython. So, please use the latest version of CircuitPython with these examples. (At least 5.0.0-beta.3)

One of the things we baked into CircuitPython is 'HID' (**H**uman **I**nterface **D**evice) control - that means keyboard and mouse capabilities. This means your CircuitPython board can act like a keyboard device and press key commands, or a mouse and have it move the mouse pointer around and press buttons. This is really handy because even if you cannot adapt your software to work with hardware, there's almost always a keyboard interface - so if you want to have a capacitive touch interface for a game, say, then keyboard emulation can often get you going really fast!

This section walks you through the code to create a keyboard or mouse emulator. First we'll go through an example that uses pins on your board to emulate keyboard input. Then, we will show you how to wire up a joystick to act as a mouse, and cover the code needed to make that happen.

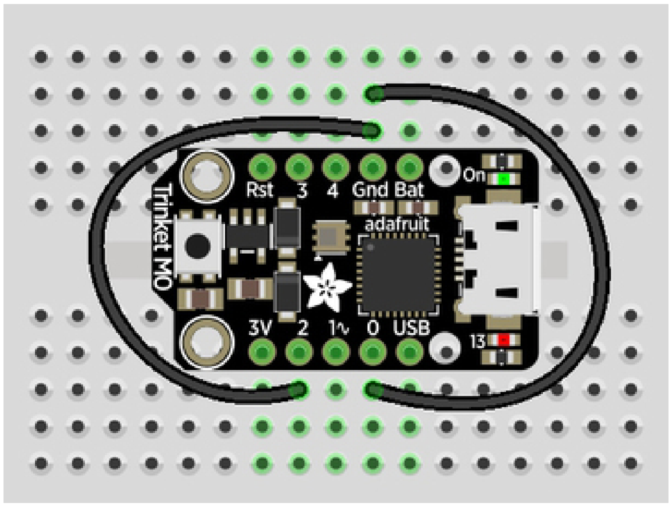
You'll need the **adafruit_hid** library folder if you don't already have it in your /lib folder! You can get it from the CircuitPython Library Bundle (https://adafru.it/y8E). If you need help installing the library, check out the CircuitPython Libraries page (https://adafru.it/ABU).

CircuitPython Keyboard Emulator

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
# CircuitPython demo - Keyboard emulator

import time

import board
import digitalio
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_hid.keycode import Keycode

# A simple neat keyboard demo in CircuitPython

# The pins we'll use, each will have an internal pullup
keypress_pins = [board.A1, board.A2]
# Our array of key objects
key_pin_array = []
# The Keycode sent for each button, will be paired with a control key
keys_pressed = [Keycode.A, "Hello World!\n"]
control_key = Keycode.SHIFT

# The keyboard object!
time.sleep(1)  # Sleep for a bit to avoid a race condition on some systems
keyboard = Keyboard(usb_hid.devices)
keyboard_layout = KeyboardLayoutUS(keyboard)  # We're in the US :)

# Make all pin objects inputs with pullups
for pin in keypress_pins:
    key_pin = digitalio.DigitalInOut(pin)
    key_pin = digitalio.DigitalInOut(pin)
    key_pin.direction = digitalio.Direction.INPUT
    key_pin.pull = digitalio.Pull.UP
    key_pin_array.append(key_pin)

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

print("Waiting for key pin...")

while True:
    # Check each pin
    for key_pin in key_pin_array:
        if not key_pin.value:  # Is it grounded?
            i = key_pin_array.index(key_pin)
            print("Pin #%d is grounded." % i)

            # Turn on the red LED
            led.value = True

            while not key_pin.value:
                pass  # Wait for it to be ungrounded!
            # "Type" the Keycode or string
            key = keys_pressed[i]  # Get the corresponding Keycode or string
            if isinstance(key, str):  # If it's a string...
                keyboard_layout.write(key)  # ...Print the string
            else:  # If it's not a string...
                keyboard.press(control_key, key)  # "Press"...
                keyboard.release_all()  # ..."Release"!

            # Turn off the red LED
            led.value = False

    time.sleep(0.01)
```

Connect pin **A1** *or* **A2** to ground, using a wire or alligator clip, then disconnect it to send the key press "A" or the string "Hello world!"



This wiring example shows A1 and A2 connected to ground.

Remember, on Trinket, A1 and A2 are labeled 2 and 0! On other boards, you will have A1 and A2 labeled as expected.

Create the Objects and Variables

First, we assign some variables for later use. We create three arrays assigned to variables: keypress_pins , key_pin_array , and keys_pressed . The first is the pins we're going to use. The second is empty because we're going to fill it later. The third is what we would like our "keyboard" to output - in this case the letter "A" and the phrase, "Hello world!". We create our last variable assigned to control_key which allows us to later apply the shift key to our keypress. We'll be using two keypresses, but you can have up to six keypresses at once.

Next keyboard and keyboard_layout objects are created. We only have US right now (if you make other layouts please submit a GitHub pull request!). The time.sleep(1) avoids an error that can happen if the program gets run as soon as the board gets plugged in, before the host computer finishes connecting to the board.

Then we take the pins we chose above, and create the pin objects, set the direction and give them each a pullup. Then we apply the pin objects to key_pin_array so we can use them later.

Next we set up the little red LED to so we can use it as a status light.

The last thing we do before we start our loop is print , "Waiting for key pin..." so you know the code is ready and waiting!

The Main Loop

Inside the loop, we check each pin to see if the state has changed, i.e. you connected the pin to ground. Once it changes, it prints, "Pin # grounded." to let you know the ground state has been detected. Then we turn on the red LED. The code waits for the state to change again, i.e. it waits for you to unground the pin by disconnecting the wire attached to the pin from ground.

Then the code gets the corresponding keys pressed from our array. If you grounded and ungrounded A1, the code retrieves the keypress a , if you grounded and ungrounded A2, the code retrieves the string, "Hello world!"

If the code finds that it's retrieved a string, it prints the string, using the keyboard_layout to determine the keypresses. Otherwise, the code prints the keypress from the control_key and the keypress "a", which result in "A". Then it calls keyboard.release_all() . You always want to call this soon after a keypress or you'll end up with a stuck key which is really annoying!

Instead of using a wire to ground the pins, you can try wiring up buttons like we did in CircuitPython Digital In & Out (https://adafru.it/Beo). Try altering the code to add more pins for more keypress options!

CircuitPython Mouse Emulator

Copy and paste the code into **code.py** using your favorite editor, and save the file.

```
import time

import analogio
import board
import digitalio
import usb_hid
from adafruit_hid.mouse import Mouse

mouse = Mouse(usb_hid.devices)

x_axis = analogio.AnalogIn(board.A0)
y_axis = analogio.AnalogIn(board.A1)
select = digitalio.DigitalInOut(board.A2)
```

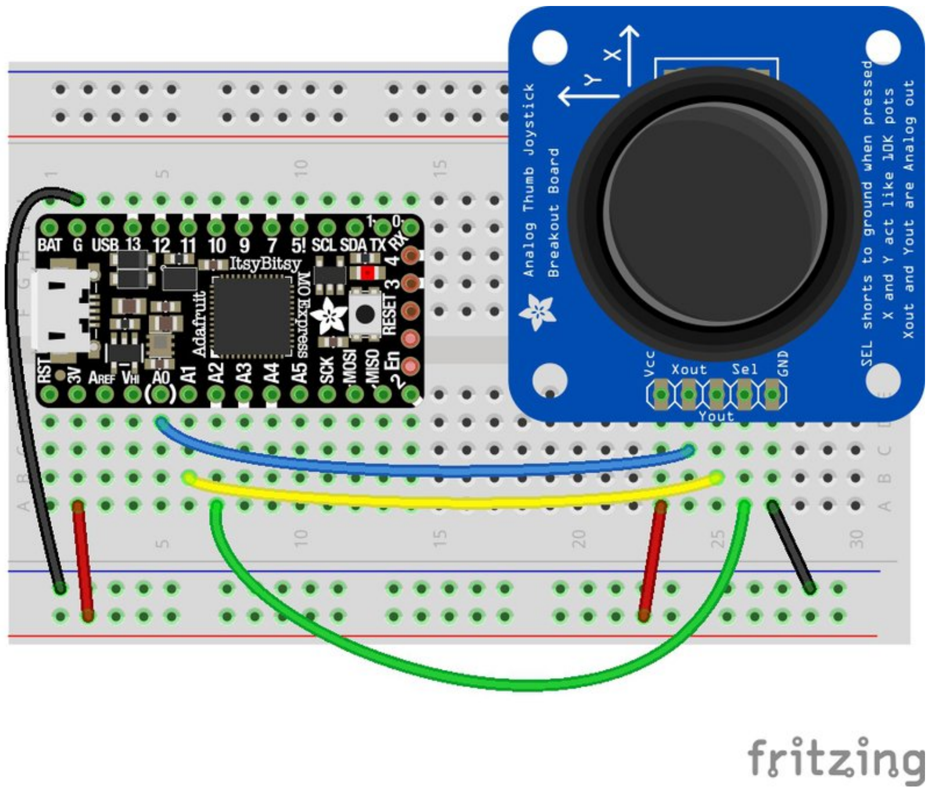
```
select.direction = digitalio.Direction.INPUT
select.pull = digitalio.Pull.UP

pot_min = 0.00
pot_max = 3.29
step = (pot_max - pot_min) / 20.0


def get_voltage(pin):
    return (pin.value * 3.3) / 65536


def steps(axis):
    """ Maps the potentiometer voltage range to 0-20 """
    return round((axis - pot_min) / step)


while True:
    x = get_voltage(x_axis)
    y = get_voltage(y_axis)

    if select.value is False:
        mouse.click(Mouse.LEFT_BUTTON)
        time.sleep(0.2)  # Debounce delay

    if steps(x) > 11.0:
        # print(steps(x))
        mouse.move(x=1)
    if steps(x) < 9.0:
        # print(steps(x))
        mouse.move(x=-1)

    if steps(x) > 19.0:
        # print(steps(x))
        mouse.move(x=8)
    if steps(x) < 1.0:
        # print(steps(x))
        mouse.move(x=-8)

    if steps(y) > 11.0:
        # print(steps(y))
        mouse.move(y=-1)
    if steps(y) < 9.0:
        # print(steps(y))
        mouse.move(y=1)

    if steps(y) > 19.0:
        # print(steps(y))
        mouse.move(y=-8)
    if steps(y) < 1.0:
        # print(steps(y))
        mouse.move(y=8)
```

For this example, we've wired up a 2-axis thumb joystick with a select button. We use this to emulate the mouse movement and the mouse left-button click. To wire up this joytick:

- Connect **VCC** on the joystick to the **3V** on your board. Connect **ground** to **ground**.
- Connect **Xout** on the joystick to pin **A0** on your board.

- Connect **Yout** on the joystick to pin **A1** on your board.
- Connect **Sel** on the joystick to pin **A2** on your board.

Remember, Trinket's pins are labeled differently. Check the Trinket Pinouts page (https://adafru.it/AMd) to verify your wiring.



To use this demo, simply move the joystick around. The mouse will move slowly if you move the joystick a little off center, and more quickly if you move it as far as it goes. Press down on the joystick to click the mouse. Awesome! Now let's take a look at the code.

Create the Objects and Variables

First we create the mouse object.

Next, we set `x_axis` and `y_axis` to pins `A0` and `A1`. Then we set `select` to `A2`, set it as input and give it a pullup.

The x and y axis on the joystick act like 2 potentiometers. We'll be using them just like we did inCircuitPython Analog In (https://adafru.it/Bep). We set `pot_min` and `pot_max` to be the minimum and maximum voltage read from the potentiometers. We assign `step = (pot_max - pot_min) / 20.0` to use in a helper function.

CircuitPython HID Mouse Helpers

First we have the `get_voltage()` helper so we can get the correct readings from the potentiometers. Look familiar? We learned about it in Analog In (https://adafru.it/Bep).

Second, we have `steps(axis)`. To use it, you provide it with the axis you're reading. This is where we're going to use the `step` variable we assigned earlier. The potentiometer range is 0-3.29. This is a small range. It's even smaller with the joystick because the joystick sits at the center of this range, 1.66, and the + and - of each axis is above and below this number. Since we need to have thresholds in our code, we're going to map that range of 0-3.29 to while numbers between 0-20.0 using this helper function. That way we can simplify our code and use larger ranges for our thresholds instead of trying to figure out tiny decimal number changes.

Main Loop

First we assign `x` and `y` to read the voltages from `x_axis` and `y_axis`.

Next, we check to see when the state of the select button is `False`. It defaults to `True` when it is not pressed, so if the state is `False`, the button has been pressed. When it's pressed, it sends the command to click the left mouse button. The `time.sleep(0.2)` prevents it from reading multiple clicks when you've only clicked once.

Then we use the `steps()` function to set our mouse movement. There are two sets of two `if` statements for each axis. Remember that `10` is the center step, as we've mapped the range `0-20`. The first set for each axis says if the joystick moves 1 step off center (left or right for the x axis and up or down for the y axis), to move the mouse the appropriate direction by 1 unit. The second set for each axis says if the joystick is moved to the lowest or highest step for each axis, to move the mouse the appropriate direction by 8 units. That way you have the option to move the mouse slowly or quickly!

To see what `step` the joystick is at when you're moving it, uncomment the `print` statements by removing the `# ` from the lines that look like `# print(steps(x))`, and connecting to the serial console to see the output. Consider only uncommenting one set at a time, or you end up with a huge amount of information scrolling very quickly, which can be difficult to read!

For more detail check out the documentation at https://circuitpython.readthedocs.io/projects/hid/en/latest/

CircuitPython Storage

CircuitPython boards show up as as USB drive, allowing you to edit code directly on the board. You've been doing this for a while. By now, maybe you've wondered, "Can I write data *from CircuitPython* to the storage drive to act as a datalogger?" The answer is **yes**!

However, it is a little tricky. You need to add some special code to **boot.py**, not just **code.py**. That's because you have to set the filesystem to be read-only when you need to edit code to the disk from your computer, and set it to writeable when you want the CircuitPython core to be able to write.

You can only have either your computer edit the CIRCUITPY drive files, or CircuitPython. You cannot have both write to the drive at the same time. (Bad Things Will Happen so we do not allow you to do it!)

The following is your new **boot.py**. Copy and paste the code into **boot.py** using your favorite editor. You may need to create a new file.

```
import board
import digitalio
import storage

# For Gemma M0, Trinket M0, Metro M0/M4 Express, ItsyBitsy M0/M4 Express
switch = digitalio.DigitalInOut(board.D2)

# For Feather M0/M4 Express
# switch = digitalio.DigitalInOut(board.D5)

# For Circuit Playground Express, Circuit Playground Bluefruit
# switch = digitalio.DigitalInOut(board.D7)

switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

# If the switch pin is connected to ground CircuitPython can write to the drive
storage.remount("/", switch.value)
```

For **Gemma M0, Trinket M0, Metro M0 Express, Metro M4 Express, ItsyBitsy M0 Express and ItsyBitsy M4 Express**, no changes to the initial code are needed.

For **Feather M0 Express and Feather M4 Express**, comment out `switch = digitalio.DigitalInOut(board.D2)`, and uncomment `switch = digitalio.DigitalInOut(board.D5)`.

For **Circuit Playground Express and Circuit Playground Bluefruit**, comment out `switch = digitalio.DigitalInOut(board.D2)`, and uncomment `switch = digitalio.DigitalInOut(board.D7)`. Remember, D7 is the onboard slide switch, so there's no extra wires or alligator clips needed.

Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

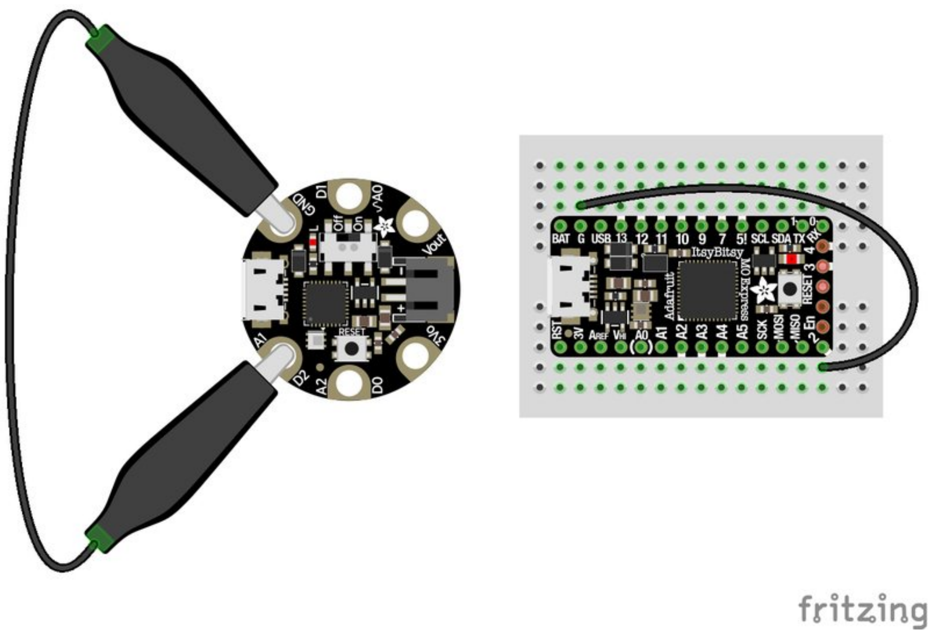
The following is your new **code.py**. Copy and paste the code into **code.py** using your favorite editor.

```
import time

import board
import digitalio
import microcontroller

led = digitalio.DigitalInOut(board.D13)
led.switch_to_output()

try:
    with open("/temperature.txt", "a") as fp:
        while True:
            temp = microcontroller.cpu.temperature
            # do the C-to-F conversion here if you would like
            fp.write('{0:f}\n'.format(temp))
            fp.flush()
            led.value = not led.value
            time.sleep(1)
except OSError as e:
    delay = 0.5
    if e.args[0] == 28:
        delay = 0.25
    while True:
        led.value = not led.value
        time.sleep(delay)
```

Logging the Temperature

The way boot.py works is by checking to see if the pin you specified in the switch setup in your code is connected to a ground pin. If it is, it changes the read-write state of the file system, so the CircuitPython core can begin logging the temperature to the board.

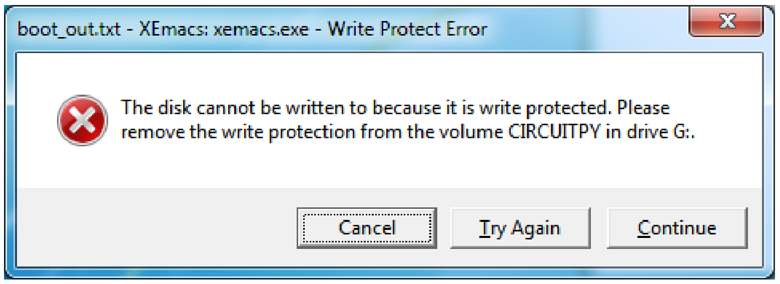
For help finding the correct pins, see the wiring diagrams and information in the Find the Pins section of the CircuitPython Digital In & Out guide (https://adafru.it/Bes). Instead of wiring up a switch, however, you'll be connecting the pin directly to ground with alligator clips or jumper wires.



boot.py only runs on first boot of the device, not if you re-load the serial console with ctrl+D or if you save a file. You must EJECT the USB drive, then physically press the reset button!

Once you copied the files to your board, eject it and unplug it from your computer. If you're using your Circuit Playground Express, all you have to do is make sure the switch is to the right. Otherwise, use alligator clips or jumper wires to connect the chosen pin to ground. Then, plug your board back into your computer.

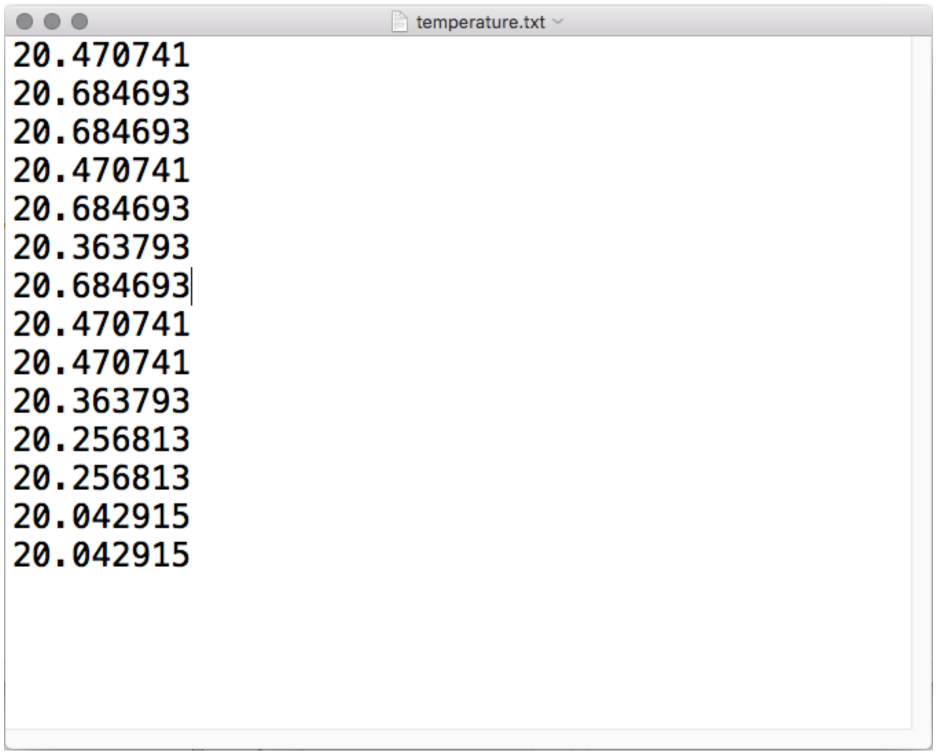
You will not be able to edit code on your CIRCUITPY drive anymore!



boot_out.txt - XEmacs: xemacs.exe - Write Protect Error

The disk cannot be written to because it is write protected. Please remove the write protection from the volume CIRCUITPY in drive G:.

The red LED should blink once a second and you will see a new**temperature.txt** file on CIRCUITPY.



```
20.470741
20.684693
20.684693
20.470741
20.684693
20.363793
20.684693
20.470741
20.470741
20.363793
20.256813
20.256813
20.042915
20.042915
```

This file gets updated once per second, but you won't see data come in live. Instead, when you're ready to grab the data, eject and unplug your board. For CPX, move the switch to the left, otherwise remove the wire connecting the pin to ground. Now it will be possible for you to write to the filesystem from your computer again, but it will not be logging data.

We have a more detailed guide on this project available here: CPU Temperature Logging with CircuitPython. (https://adafru.it/zuF) If you'd like more details, check it out!

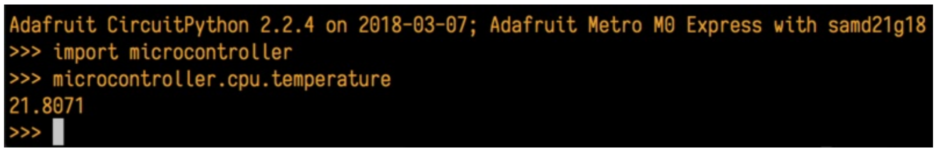
CircuitPython CPU Temp

There is a CPU temperature sensor built into every ATSAMD21, ATSAMD51 and nRF52840 chips. CircuitPython makes it really simple to read the data from this sensor. This works on the Adafruit CircuitPython boards it's built into the microcontroller used for these boards.

The data is read using two simple commands. We're going to enter them in the REPL. Plug in your board, connect to the serial console (https://adafru.it/Bec), and enter the REPL (https://adafru.it/Awz). Then, enter the following commands into the REPL:

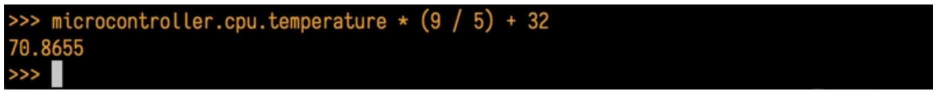
```
import microcontroller
microcontroller.cpu.temperature
```

That's it! You've printed the temperature in Celsius to the REPL. Note that it's not exactly the ambient temperature and it's not super precise. But it's close!



```
Adafruit CircuitPython 2.2.4 on 2018-03-07; Adafruit Metro M0 Express with samd21g18
>>> import microcontroller
>>> microcontroller.cpu.temperature
21.8071
>>> 
```

If you'd like to print it out in Fahrenheit, use this simple formula: Celsius * (9/5) + 32. It's super easy to do math using CircuitPython. Check it out!



```
>>> microcontroller.cpu.temperature * (9 / 5) + 32
70.8655
>>> 
```

Note that the temperature sensor built into the nRF52840 has a resolution of 0.25 degrees Celsius, so any temperature you print out will be in 0.25 degree increments.

CircuitPython Expectations

As we continue to develop CircuitPython and create new releases, we will stop supporting older releases. Visit https://circuitpython.org/downloads to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit https://circuitpython.org/libraries to download the latest Library Bundle.

Always Run the Latest Version of CircuitPython and Libraries

As we continue to develop CircuitPython and create new releases, we will stop supporting older releases. **You need to update to the latest CircuitPython (https://adafru.it/Em8).**

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. **Please update CircuitPython and then download the latest bundle (https://adafru.it/ENC).**

As we release new versions of CircuitPython, we will stop providing the previous bundles as automatically created downloads on the Adafruit CircuitPython Library Bundle repo. If you must continue to use an earlier version, you can still download the appropriate version of `mpy-cross` from the particular release of CircuitPython on the CircuitPython repo and create your own compatible .mpy library files. **However, it is best to update to the latest for both CircuitPython and the library bundle.**

I have to continue using CircuitPython 3.x or 2.x, where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 2.x and 3.x library bundles. We highly encourage you to update CircuitPython to the latest version (https://adafru.it/Em8) and use the current version of the libraries (https://adafru.it/ENC). However, if for some reason you cannot update, you can find the last available 2.x build here (https://adafru.it/FJA) and the last available 3.x build here (https://adafru.it/FJB).

Switching Between CircuitPython and Arduino

Many of the CircuitPython boards also run Arduino. But how do you switch between the two? Switching between CircuitPython and Arduino is easy.

If you're currently running Arduino and would like to start using CircuitPython, follow the steps found inWelcome to CircuitPython: Installing CircuitPython (https://adafru.it/Amd).

If you're currently running CircuitPython and would like to start using Arduino, plug in your board, and then load your Arduino sketch. If there are any issues, you can double tap the reset button to get into the bootloader and then try loading your sketch. Always backup any files you're using with CircuitPython that you want to save as they could be deleted.

That's it! It's super simple to switch between the two.

The Difference Between Express And Non-Express Boards

We often reference "Express" and "Non-Express" boards when discussing CircuitPython. What does this mean?

Express refers to the inclusion of an extra 2MB flash chip on the board that provides you with extra space for

CircuitPython and your code. This means that we're able to include more functionality in CircuitPython and you're able to do more with your code on an Express board than you would on a non-Express board.

Express boards include Circuit Playground Express, ItsyBitsy M0 Express, Feather M0 Express, Metro M0 Express and Metro M4 Express.

Non-Express boards include Trinket M0, Gemma M0, Feather M0 Basic, and other non-Express Feather M0 variants.

Non-Express Boards: Gemma and Trinket

CircuitPython runs nicely on the Gemma M0 or Trinket M0 but there are some constraints

Small Disk Space

Since we use the internal flash for disk, and that's shared with runtime code, its limited! Only about 50KB of space.

No Audio or NVM

Part of giving up that FLASH for disk means we couldn't fit everything in. There is, at this time, no support for hardware audio playpack or NVM 'eeprom'. Modules `audioio` and `bitbangio` are not included. For that support, check out the Circuit Playground Express or other Express boards.

However, I2C, UART, capacitive touch, NeoPixel, DotStar, PWM, analog in and out, digital IO, logging storage, and HID do work! Check the CircuitPython Essentials for examples of all of these.

Differences Between CircuitPython and MicroPython

For the differences between CircuitPython and MicroPython, check out the CircuitPython documentation (https://adafru.it/Bvz).

Differences Between CircuitPython and Python

Python (also known as CPython) is the language that MicroPython and CircuitPython are based on. There are many similarities, but there are also many differences. This is a list of a few of the differences.

Python Libraries

Python is advertised as having "batteries included", meaning that many standard libraries are included. Unfortunately, for space reasons, many Python libraries are not available. So for instance while we wish you could `import numpy`, `numpy` isn't available. So you may have to port some code over yourself!

Integers in CircuitPython

On the non-Express boards, integers can only be up to 31 bits long. Integers of unlimited size are not supported. The largest positive integer that can be represented is 2^{30}-1, 1073741823, and the most negative integer possible is -2^{30}, -1073741824.

As of CircuitPython 3.0, Express boards have arbitrarily long integers as in Python.

Floating Point Numbers and Digits of Precision for Floats in CircuitPython

Floating point numbers are single precision in CircuitPython (not double precision as in Python). The largest floating point magnitude that can be represented is about +/-3.4e38. The smallest magnitude that can be represented with full accuracy is about +/-1.7e-38, though numbers as small as +/-5.6e-45 can be represented with reduced accuracy.

CircuitPython's floats have 8 bits of exponent and 22 bits of mantissa (not 24 like regular single precision floating point), which is about five or six decimal digits of precision.

Differences between MicroPython and Python

For a more detailed list of the differences between CircuitPython and Python, you can look at the MicroPython documentation. We keep up with MicroPython stable releases, so check out the core 'differences' they document here. (https://adafru.it/zwA)

CircuitPython Libraries and Drivers

CircuitPython Libraries and Drivers (https://adafru.it/AYD)

CircuitPython Libraries

We have tons of CircuitPython libraries that can be used by microcontroller boards or single board computers such as Raspberry Pi. Here's a quick listing that is automatically generated

Adafruit CircuitPython Libraries

Here is a listing of current Adafruit CircuitPython Libraries. There are 224 libraries available.

Drivers:

- Adafruit CircuitPython 74HC595 (PyPi)
- Adafruit CircuitPython ADS1x15 (PyPi)
- Adafruit CircuitPython ADT7410 (PyPi)
- Adafruit CircuitPython ADXL34x (PyPi)
- Adafruit CircuitPython AM2320 (PyPi)
- Adafruit CircuitPython AMG88xx (PyPi)
- Adafruit CircuitPython APDS9960 (PyPi)
- Adafruit CircuitPython AS726x (PyPi)
- Adafruit CircuitPython ATECC (PyPi)
- Adafruit CircuitPython BD3491FS (PyPi)
- Adafruit CircuitPython BME280 (PyPi)
- Adafruit CircuitPython BME680 (PyPi)
- Adafruit CircuitPython BMP280 (PyPi)
- Adafruit CircuitPython BMP3XX (PyPi)
- Adafruit CircuitPython BNO055 (PyPi)
- Adafruit CircuitPython BluefruitSPI (PyPi)
- Adafruit CircuitPython CAP1188 (PyPi)

- Adafruit CircuitPython CCS811 (PyPi)
- Adafruit CircuitPython CLUE
- Adafruit CircuitPython CharLCD (PyPi)
- Adafruit CircuitPython CircuitPlayground
- Adafruit CircuitPython Crickit (PyPi)
- Adafruit CircuitPython DHT (PyPi)
- Adafruit CircuitPython DPS310 (PyPi)
- Adafruit CircuitPython DRV2605 (PyPi)
- Adafruit CircuitPython DS1307 (PyPi)
- Adafruit CircuitPython DS1841 (PyPi)
- Adafruit CircuitPython DS18X20 (PyPi)
- Adafruit CircuitPython DS2413 (PyPi)
- Adafruit CircuitPython DS3231 (PyPi)
- Adafruit CircuitPython DS3502 (PyPi)
- Adafruit CircuitPython DisplayIO SSD1305 (PyPi)
- Adafruit CircuitPython DisplayIO SSD1306
- Adafruit CircuitPython DotStar (PyPi)
- Adafruit CircuitPython DymoScale (PyPi)
- Adafruit CircuitPython EPD (PyPi)
- Adafruit CircuitPython ESP ATcontrol (PyPi)
- Adafruit CircuitPython ESP32SPI (PyPi)
- Adafruit CircuitPython FRAM (PyPi)
- Adafruit CircuitPython FXAS21002C (PyPi)
- Adafruit CircuitPython FXOS8700 (PyPi)
- Adafruit CircuitPython Fingerprint (PyPi)
- Adafruit CircuitPython FocalTouch (PyPi)
- Adafruit CircuitPython GPS (PyPi)
- Adafruit CircuitPython HCSR04 (PyPi)
- Adafruit CircuitPython HT16K33 (PyPi)
- Adafruit CircuitPython HTS221 (PyPi)
- Adafruit CircuitPython HTU21D (PyPi)
- Adafruit CircuitPython HX8357
- Adafruit CircuitPython ICM20649 (PyPi)
- Adafruit CircuitPython IL0373 (PyPi)
- Adafruit CircuitPython IL0398 (PyPi)
- Adafruit CircuitPython IL91874 (PyPi)
- Adafruit CircuitPython ILI9341
- Adafruit CircuitPython INA219 (PyPi)
- Adafruit CircuitPython INA260 (PyPi)
- Adafruit CircuitPython IRRemote (PyPi)
- Adafruit CircuitPython IS31FL3731 (PyPi)
- Adafruit CircuitPython L3GD20 (PyPi)
- Adafruit CircuitPython LIDARLite (PyPi)
- Adafruit CircuitPython LIS2MDL (PyPi)
- Adafruit CircuitPython LIS3DH (PyPi)
- Adafruit CircuitPython LIS3MDL (PyPi)
- Adafruit CircuitPython LPS2X (PyPi)
- Adafruit CircuitPython LPS35HW (PyPi)
- Adafruit CircuitPython LSM303 Accel (PyPi)
- Adafruit CircuitPython LSM303DLH Mag (PyPi)
- Adafruit CircuitPython LSM303 (PyPi)

- Adafruit CircuitPython LSM6DS (PyPi)
- Adafruit CircuitPython LSM9DS0 (PyPi)
- Adafruit CircuitPython LSM9DS1 (PyPi)
- Adafruit CircuitPython MAX31855 (PyPi)
- Adafruit CircuitPython MAX31856 (PyPi)
- Adafruit CircuitPython MAX31865 (PyPi)
- Adafruit CircuitPython MAX7219 (PyPi)
- Adafruit CircuitPython MAX9744 (PyPi)
- Adafruit CircuitPython MCP230xx (PyPi)
- Adafruit CircuitPython MCP3xxx (PyPi)
- Adafruit CircuitPython MCP4725 (PyPi)
- Adafruit CircuitPython MCP4728 (PyPi)
- Adafruit CircuitPython MCP9600 (PyPi)
- Adafruit CircuitPython MCP9808 (PyPi)
- Adafruit CircuitPython MLX90393 (PyPi)
- Adafruit CircuitPython MLX90614 (PyPi)
- Adafruit CircuitPython MLX90640 (PyPi)
- Adafruit CircuitPython MMA8451 (PyPi)
- Adafruit CircuitPython MPL115A2 (PyPi)
- Adafruit CircuitPython MPL3115A2 (PyPi)
- Adafruit CircuitPython MPR121 (PyPi)
- Adafruit CircuitPython MPRLS (PyPi)
- Adafruit CircuitPython MPU6050 (PyPi)
- Adafruit CircuitPython MSA301 (PyPi)
- Adafruit CircuitPython MatrixKeypad (PyPi)
- Adafruit CircuitPython NeoPixel SPI (PyPi)
- Adafruit CircuitPython NeoPixel (PyPi)
- Adafruit CircuitPython NeoTrellis (PyPi)
- Adafruit CircuitPython Nunchuk
- Adafruit CircuitPython PCA9685 (PyPi)
- Adafruit CircuitPython PCD8544 (PyPi)
- Adafruit CircuitPython PCF8523 (PyPi)
- Adafruit CircuitPython PCT2075 (PyPi)
- Adafruit CircuitPython PN532 (PyPi)
- Adafruit CircuitPython Pixie (PyPi)
- Adafruit CircuitPython PyPortal (PyPi)
- Adafruit CircuitPython RA8875 (PyPi)
- Adafruit CircuitPython RFM69 (PyPi)
- Adafruit CircuitPython RFM9x (PyPi)
- Adafruit CircuitPython RGB Display (PyPi)
- Adafruit CircuitPython RPLIDAR (PyPi)
- Adafruit CircuitPython RockBlock (PyPi)
- Adafruit CircuitPython SD (PyPi)
- Adafruit CircuitPython SGP30 (PyPi)
- Adafruit CircuitPython SHT31D (PyPi)
- Adafruit CircuitPython SI4713 (PyPi)
- Adafruit CircuitPython SI5351 (PyPi)
- Adafruit CircuitPython SI7021 (PyPi)
- Adafruit CircuitPython SSD1305 (PyPi)
- Adafruit CircuitPython SSD1306 (PyPi)
- Adafruit CircuitPython SSD1322

- Adafruit CircuitPython SSD1325 (PyPi)
- Adafruit CircuitPython SSD1327
- Adafruit CircuitPython SSD1331
- Adafruit CircuitPython SSD1351
- Adafruit CircuitPython SSD1608
- Adafruit CircuitPython SSD1675 (PyPi)
- Adafruit CircuitPython ST7735R
- Adafruit CircuitPython ST7735
- Adafruit CircuitPython ST7789
- Adafruit CircuitPython STMPE610 (PyPi)
- Adafruit CircuitPython Seesaw (PyPi)
- Adafruit CircuitPython SharpMemoryDisplay (PyPi)
- Adafruit CircuitPython TCA9548A (PyPi)
- Adafruit CircuitPython TCS34725 (PyPi)
- Adafruit CircuitPython TFmini (PyPi)
- Adafruit CircuitPython TLC5947 (PyPi)
- Adafruit CircuitPython TLC59711 (PyPi)
- Adafruit CircuitPython TLV493D (PyPi)
- Adafruit CircuitPython TMP006 (PyPi)
- Adafruit CircuitPython TMP007 (PyPi)
- Adafruit CircuitPython TPA2016 (PyPi)
- Adafruit CircuitPython TSL2561 (PyPi)
- Adafruit CircuitPython TSL2591 (PyPi)
- Adafruit CircuitPython Thermal Printer (PyPi)
- Adafruit CircuitPython Thermistor (PyPi)
- Adafruit CircuitPython Touchscreen (PyPi)
- Adafruit CircuitPython TrellisM4 (PyPi)
- Adafruit CircuitPython Trellis (PyPi)
- Adafruit CircuitPython US100 (PyPi)
- Adafruit CircuitPython VC0706 (PyPi)
- Adafruit CircuitPython VCNL4010 (PyPi)
- Adafruit CircuitPython VCNL4040 (PyPi)
- Adafruit CircuitPython VEML6070 (PyPi)
- Adafruit CircuitPython VEML6075 (PyPi)
- Adafruit CircuitPython VEML7700 (PyPi)
- Adafruit CircuitPython VL53L0X (PyPi)
- Adafruit CircuitPython VL6180X (PyPi)
- Adafruit CircuitPython VS1053 (PyPi)
- Adafruit CircuitPython WS2801 (PyPi)
- Adafruit CircuitPython Wiznet5k (PyPi)

Helpers:

- Adafruit CircuitPython AVRprog (PyPi)
- Adafruit CircuitPython AWS IOT (PyPi)
- Adafruit CircuitPython AdafruitIO (PyPi)
- Adafruit CircuitPython AzureIoT (PyPi)
- Adafruit CircuitPython BLE Apple Media (PyPi)
- Adafruit CircuitPython BLE Apple Notification Center (PyPi)
- Adafruit CircuitPython BLE BroadcastNet (PyPi)
- Adafruit CircuitPython BLE Cycling Speed and Cadence (PyPi)
- Adafruit CircuitPython BLE Eddystone (PyPi)

- Adafruit CircuitPython BLE Heart Rate (PyPi)
- Adafruit CircuitPython BLE Magic Light
- Adafruit CircuitPython BLE Radio (PyPi)
- Adafruit CircuitPython BLE iBBQ
- Adafruit CircuitPython BLE (PyPi)
- Adafruit CircuitPython Bitmap Font (PyPi)
- Adafruit CircuitPython BitmapSaver (PyPi)
- Adafruit CircuitPython BluefruitConnect (PyPi)
- Adafruit CircuitPython BoardTest
- Adafruit CircuitPython BusDevice (PyPi)
- Adafruit CircuitPython CursorControl (PyPi)
- Adafruit CircuitPython Debouncer
- Adafruit CircuitPython Debug I2C
- Adafruit CircuitPython Display Button (PyPi)
- Adafruit CircuitPython Display Notification (PyPi)
- Adafruit CircuitPython Display Shapes (PyPi)
- Adafruit CircuitPython Display Text (PyPi)
- Adafruit CircuitPython FancyLED (PyPi)
- Adafruit CircuitPython FeatherWing (PyPi)
- Adafruit CircuitPython GC IOT Core (PyPi)
- Adafruit CircuitPython Gizmo (PyPi)
- Adafruit CircuitPython HID (PyPi)
- Adafruit CircuitPython Hue (PyPi)
- Adafruit CircuitPython ImageLoad
- Adafruit CircuitPython IterTools
- Adafruit CircuitPython JWT (PyPi)
- Adafruit CircuitPython LED Animation
- Adafruit CircuitPython LIFX (PyPi)
- Adafruit CircuitPython Logging
- Adafruit CircuitPython MIDI (PyPi)
- Adafruit CircuitPython MiniMQTT (PyPi)
- Adafruit CircuitPython MotorKit (PyPi)
- Adafruit CircuitPython Motor (PyPi)
- Adafruit CircuitPython NTP (PyPi)
- Adafruit CircuitPython OneWire (PyPi)
- Adafruit CircuitPython PYOA
- Adafruit CircuitPython ProgressBar (PyPi)
- Adafruit CircuitPython PyBadger (PyPi)
- Adafruit CircuitPython Pypixelbuf (PyPi)
- Adafruit CircuitPython RGBLED (PyPi)
- Adafruit CircuitPython RSA (PyPi)
- Adafruit CircuitPython RTTTL (PyPi)
- Adafruit CircuitPython Register (PyPi)
- Adafruit CircuitPython Requests (PyPi)
- Adafruit CircuitPython ServoKit (PyPi)
- Adafruit CircuitPython SimpleIO (PyPi)
- Adafruit CircuitPython Slideshow (PyPi)
- Adafruit CircuitPython TinyLoRa (PyPi)
- Adafruit CircuitPython WSGI (PyPi)
- Adafruit CircuitPython Waveform (PyPi)
- Adafruit CircuitPython binascii (PyPi)

- Adafruit CircuitPython framebuf (PyPi)
- Adafruit CircuitPython hashlib (PyPi)
- Adafruit CircuitPython miniQR (PyPi)
- Adafruit CircuitPython miniesptool (PyPi)
- Adafruit CircuitPython turtle (PyPi)